

***PROGRAMMING
AND
BUILDING***

VAXELN SYSTEMS

Student Guide

DIGITAL EDUCATIONAL SERVICES
READING ENGLAND

**PROGRAMMING
AND
BUILDING**

VAXELN SYSTEMS

Student Guide

DIGITAL EDUCATIONAL SERVICES
READING ENGLAND

First Edition, March 1987

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

Copyright © 1987 by Digital Equipment Co. Ltd.

All rights reserved

Printed in U.K.

CONTENTS

CHAPTER 0	INTRODUCTION TO COURSE	
0.1	COURSE OBJECTIVES	0-1
0.2	BEFORE ATTENDING THIS COURSE	0-1
0.3	DOCUMENTATION FOR THE COURSE	0-2
0.4	OBJECTIVES OF THIS STUDENT GUIDE	0-2
0.5	ABBREVIATIONS USED IN THIS HANDOUT	0-2
0.6	CONVENTIONS USED IN THIS HANDOUT	0-3
CHAPTER 1	VAXELN CAPABILITIES AND APPLICATIONS	
1.1	THE NATURE AND PURPOSE OF VAXELN	1-1
1.2	VAXELN SYSTEM COMPONENTS	1-1
1.3	FUNCTION OF THE VAXELN KERNEL	1-2
1.4	A SIMPLE VAXELN APPLICATION	1-3
1.5	DEVELOPING A SIMPLE APPLICATION	1-5
CHAPTER 2	VAXELN EXTENSIONS TO ISO PASCAL	
2.1	SOURCE TEXT	2-1
2.2	DATA TYPES	2-5
2.3	ATTRIBUTES FOR DATA SIZING	2-10
2.4	CONSTANTS	2-11
2.5	INITIALIZERS	2-12
2.6	PREDECLARED NAMED CONSTANTS	2-13
2.7	PREDECLARED ENUMERATED TYPES	2-14
2.8	VARIABLES	2-15
2.9	ALLOCATION OF STORAGE TO CONSTANTS AND VARIABLES	2-18
2.10	INITIALIZATION OF SHARED DATA	2-20
2.11	EXPRESSIONS AND OPERATORS	2-20
2.12	CONCATENATING STRINGS	2-21
2.13	STATEMENTS IN VAXELN PASCAL	2-21
2.14	PROCEDURES AND FUNCTIONS	2-22
2.15	QUEUES	2-23
CHAPTER 3	KERNEL OBJECTS AND THEIR USE	
3.1	WHAT ARE KERNEL OBJECTS?	3-1
CHAPTER 4	PROGRAM DEVELOPMENT	
4.1	COMPILING VAXELN SOURCES	4-1
4.2	USING THE DCL COMMAND LINK	4-4

CHAPTER 5	SYSTEM DEVELOPMENT	
5.1	THE EBUILD COMMAND	5-1
CHAPTER 6	BOOTING AND DOWNLINE LOADING	
6.1	BOOTING FROM DISK	6-1
6.2	DOWNLINE LOADING	6-2
6.3	CONFIGURING THE HOST NETWORK DATA BASE	6-4
6.4	CONFIGURING BOOTSTRAP LOADERS	6-4
6.5	DOWNLINE LOADING	6-5
6.6	RELOADING TARGETS THAT HAVE NETWORK SERVICE	6-6
6.7	MONITORING NETWORK EVENTS	6-6
6.8	PROBLEMS WITH DOWNLINE LOADING	6-7
6.9	PROBLEM BOOTING FROM DEQNA	6-8
CHAPTER 7	DEBUGGING	
7.1	METHODS OF DEBUGGING	7-1
7.2	CHOOSING A DEBUGGING MODE	7-2
7.3	KERNEL DEBUGGING	7-2
7.4	PROGRAM DEVELOPMENT AND DEBUGGING	7-2
7.5	EDEBUG COMMAND AND ITS QUALIFIERS	7-2
7.6	LOGICAL NAMES AND EDEBUG	7-3
7.7	EDEBUG FACILITIES	7-3
7.8	CONTROL-C SESSION	7-4
7.9	EDEBUG COMMANDS	7-4
7.10	SESSION LOGGING	7-7
CHAPTER 8	PROCESSES, JOBS AND PROGRAM STRUCTURE	
8.1	JOB DEFINITION	8-1
8.2	PROCESS DEFINITION	8-1
8.3	PROGRAM DEFINITION	8-1
8.4	ROUTINES IN VAXELN	8-2
8.5	COMPILATION UNIT	8-3
8.6	THE MODULE IN VAXELN	8-3
8.7	PROCESS STATES	8-4
8.8	PROCESS PRIORITIES	8-4
8.9	PROCESSES AND MEMORY MANAGEMENT	8-4
CHAPTER 9	TECHNIQUES OF SYNCHRONIZATION	
9.1	INTRODUCTION	9-1
9.2	WAIT PROCEDURES	9-2
9.3	SATISFYING A WAIT FOR KERNEL OBJECTS	9-3
9.4	PROCEDURES FOR SYNCHRONIZATION	9-4
9.5	THE MUTEX	9-5
9.6	SPECIFYING ABSOLUTE AND DELTA TIMES	9-6

CHAPTER 10 COMMUNICATION BETWEEN JOBS

10.1	INTRODUCTION	10-1
10.2	COMMUNICATION WITHIN A SINGLE JOB ON A SINGLE VAXELN NODE	10-2
10.3	COMMUNICATION BETWEEN MULTIPLE JOBS ON A SINGLE VAXELN NODE	10-4
10.4	MESSAGE PORTS	10-7
10.5	SENDING AND RECEIVING MESSAGES	10-9
10.6	MESSAGE TRANSMISSION METHODS	10-10
10.7	HINTS ON COMMUNICATION	10-15

CHAPTER 11 NETWORK FACILITIES

11.1	LOCAL AND REMOTE LINKS WITH CIRCUITS	11-2
11.2	DECNET MESSAGE SIZE	11-2
11.3	NAME SERVERS	11-5
11.4	NODE IDENTIFICATION	11-6
11.5	MANAGING YOUR VAXELN NETWORK	11-8
11.6	NCP COMMANDS SUPPORTED BY THE VAXELN NML	11-9
11.7	CONNECTIONS TO AND FROM VMS	11-10

CHAPTER 12 USING FILES

12.1	FILE ROUTINES	12-1
12.2	FILE OPENING AND CLOSING	12-2
12.3	INTERNAL AND EXTERNAL FILES	12-3
12.4	FILE TYPES	12-3
12.5	FILE ACCESS	12-3
12.6	RECORD TYPES	12-3
12.7	ENUMERATED TYPES USED IN FILE HANDLING	12-4
12.8	FILE AND DEVICE SPECIFICATIONS	12-4
12.9	REMOTE FILE ACCESS	12-7

CHAPTER 13 DEVICES, DRIVERS AND INTERRUPTS

13.1	INTRODUCTION	13-1
13.2	DEFINITIONS	13-1
13.3	DRIVERS SUPPLIED WITH VAXELN	13-2
13.4	INTERRUPT SERVICE ROUTINES	13-2
13.5	DISABLING INTERRUPTS	13-4
13.6	INTERRUPT PRIORITY LEVELS	13-4
13.7	UNEXPECTED INTERRUPTS - POWER FAILURE	13-5
13.8	POWERFAIL RECOVERY ROUTINES	13-5
13.9	DEVICE INITIALIZATION ROUTINES	13-7

CHAPTER 14 SYSTEM SECURITY

14.1	INTRODUCTION	14-1
14.2	DEFAULT PROTECTION ON VAXELN	14-2
14.3	VAXELN SYSTEM SOFTWARE PROTECTION	14-2

14.4	AUTHORIZATION FACILITIES	14-2
14.5	SPECIFYING REMOTE DESTINATIONS	14-3
14.6	INCLUDING AUTHORIZATION IN A SYSTEM	14-4
14.7	USING AND MAINTAINING THE AUTHORIZATION SERVICE	14-6
14.8	FILE SECURITY FACILITIES	14-6

CHAPTER 15 HANDLING EXCEPTIONS

15.1	INTRODUCTION	15-1
15.2	DEFINITIONS	15-1
15.3	PROGRAM/PROGRAMMER RESPONSE TO EXCEPTIONS	15-2
15.4	CONDITION VALUES	15-3
15.5	THE SEVERITY FIELD IN CONDITION VALUES	15-4
15.6	STATUS CHECKING	15-4
15.7	EVENTS WHEN EXCEPTIONS ARE RAISED	15-5
15.8	THE STACK	15-5
15.9	STACK AFTER EXCEPTION OCCURS	15-11
15.10	POSSIBLE ACTION OF AN EXCEPTION HANDLER	15-12
15.11	EXCEPTIONS DURING EXCEPTIONS	15-13
15.12	ROUTINES FOR EXCEPTION HANDLING	15-14
15.13	MESSAGE FACILITIES	15-14

APPENDIX A ASCII CHARACTER CODES

APPENDIX B EXAMPLES OF LANGUAGE EXTENSIONS

B.1	INTRODUCTION	B-1
B.2	EXAMPLE OF IDENT - SIMPLE.PAS	B-1
B.3	EXAMPLE OF %INCLUDE - INCLUDE_SOURCE.PAS	B-5
B.4	EXAMPLE OF IMPORT AND EXPORT - IMPORT_1.PAS	B-8
B.5	EXAMPLE OF INCLUDE - INCLUDE_1.PAS	B-13
B.6	EXAMPLE OF RADIX SPECIFIERS - RADIX_1.PAS	B-18
B.7	EXAMPLE OF SIZE ATTRIBUTES - ATTRIBS_1.PAS	B-20
B.8	EXAMPLE OF TYPECASTING - TYPECAST_1.PAS	B-23
B.9	EXAMPLE OF SCOPE - SCOPE_1.PAS	B-27
B.10	EXAMPLE OF SCOPE - SCOPE_2.PAS	B-29
B.11	EXAMPLE OF SCOPE - SCOPE_3.PAS	B-31
B.12	EXAMPLE OF SCOPE - SCOPE_3A.PAS	B-33
B.13	EXAMPLE OF SEPARATE - SEPARATE_PROG_1.PAS	B-35
B.14	EXAMPLE OF AGGREGATE INITIALIZATION - AGGREG_1.PAS	B-39

APPENDIX C EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

C.1	EXAMPLE OF MODULE, PROGRAM AND PROCESS - MOD-PROG-PROC-1.PAS	C-1
C.2	EXAMPLE OF MODULE, PROGRAM AND PROCESS - MOD-PROG-PROC-1A.PAS	C-6
C.3	EXAMPLE OF ERROR REPORTING - ERRORS_1.PAS	C-10
C.4	EXAMPLE OF ERROR REPORTING - ERRORS_2.PAS	C-13

APPENDIX D EXAMPLES OF SYNCHRONIZATION AND TIME

D.1	EXAMPLE OF VAXELN TIME ROUTINES - TIME_1.PAS . . .	D-1
D.2	EXAMPLE OF VAXELN RUNNING FORTRAN - TIME_2.PAS . .	D-7
D.3	EXAMPLE OF VAXELN RUNNING FORTRAN - TIME_2A.PAS	D-15
D.4	EXAMPLE OF VAXELN CALLING VMS FOR THE TIME - TIME_3.PAS	D-19
D.5	EXAMPLE OF USING VAXELN INTERNAL TIME VALUES - TIME_4.PAS	D-25
D.6	EXAMPLE OF USING WAIT ROUTINE - SYNCH_1.PAS . .	D-29
D.7	EXAMPLE OF WAITING FOR AN EVENT - SYNCH_2.PAS .	D-32
D.8	EXAMPLE OF WAITING FOR OBJECTS - SYNCH_3.PAS . .	D-36
D.9	EXAMPLE OF WAITING FOR A PROCESS - SYNCH_4.PAS .	D-41
D.10	EXAMPLE OF MUTEX - SYNCH_5.PAS	D-45
D.11	EXAMPLE OF MUTEX - SYNCH_6.PAS	D-52

APPENDIX E EXAMPLES OF COMMUNICATION TECHNIQUES

E.1	EXAMPLE OF DATAGRAM COMMUNICATION - COMM_1.PAS . .	E-1
E.2	EXAMPLE OF DATAGRAM COMMUNICATION - COMM_2.PAS . .	E-7
E.3	EXAMPLE OF DATAGRAM COMMUNICATION - COMM_3.PAS .	E-14
E.4	EXAMPLE OF DATAGRAM COMMUNICATION - COMM_4.PAS .	E-23
E.5	EXAMPLE OF AREA DATA SHARING - COMM_5.PAS . . .	E-28
E.6	EXAMPLE OF DATA PASSING BY JOB ARGUMENT - COMM_6.PAS	E-34
E.7	EXAMPLE OF USING CIRCUITS - COMM_7.PAS	E-39
E.8	EXAMPLE OF USING CIRCUITS - COMM_8.PAS	E-46
E.9	EXAMPLE OF USING CIRCUITS AND EXPEDITED MESSAGES - COMM_9.PAS	E-56

APPENDIX F NETWORK EXAMPLE

F.1	EXAMPLE OF NETWORK USE - NET_1.PAS	F-1
-----	--	-----

APPENDIX G FILE HANDLING

G.1	EXAMPLE OF FILE ACCESS ROUTINES - FILE_1.PAS . . .	G-1
G.2	EXAMPLE OF REMOTE FILE HANDLING - FILE_2.FOR . .	G-6

APPENDIX H DEVICE DRIVER EXAMPLES

H.1	EXAMPLE OF SUPPLIED DRIVER - ELN\$:LPVDRIVER.PAS .	H-1
-----	--	-----

APPENDIX I EXCEPTION HANDLING

I.1	EXAMPLE OF EXCEPTION HANDLING - EXCEPT_1.PAS . . .	I-1
-----	--	-----

APPENDIX J EXERCISES

APPENDIX K SOLUTIONS TO EXERCISES

K.1	SOLUTION TO EXERCISE 1 - LAB_1.PAS	K-1
K.2	SOLUTION TO EXERCISE 1 - LAB_1_F.FOR	K-7
K.3	SOLUTION TO EXERCISE 1 - LAB_1_C.C	K-14
K.4	SOLUTION TO EXERCISE 2 - LAB_2.PAS	K-20
K.5	SOLUTION TO EXERCISE 2 - LAB_2A.PAS	K-27
K.6	SOLUTION TO EXERCISE 2 - LAB_2A_TDEFS.PAS . . .	K-31
K.7	SOLUTION TO EXERCISE 2 - LAB_2B.PAS	K-34
K.8	SOLUTION TO EXERCISE 2 - LAB_2B_DEFS.PAS . . .	K-37
K.9	SOLUTION TO EXERCISE 2 - LAB_2_C.C	K-40
K.10	SOLUTION TO EXERCISE 3 - LAB_3.PAS	K-47
K.11	SOLUTION TO EXERCISE 3 - LAB_3A.PAS	K-50
K.12	SOLUTION TO EXERCISE 4 - LAB_4.PAS	K-56
K.13	SOLUTION TO EXERCISE 4 - LAB_4A.PAS	K-58
K.14	SOLUTION TO EXERCISE 4 - LAB_4B.PAS	K-62
K.15	SOLUTION TO EXERCISE 5 - LAB_5.PAS	K-67
K.16	SOLUTION TO EXERCISE 5 - LAB_5A.PAS	K-72
K.17	SOLUTION TO EXERCISE 6 - LAB_6.PAS	K-78
K.18	SOLUTION TO EXERCISE 6 - LAB_6A.PAS	K-81

ACKNOWLEDGEMENT

The author wishes to acknowledge that some of the programs dealing with real-time issues came from a course developed by the Real-time Product Support Unit of the Atlanta Customer Support Center.

DEC Educational Services
Reading, England
March 1987

CHAPTER 0

INTRODUCTION TO COURSE

0.1 COURSE OBJECTIVES

On completion of the course students should be able to:-

1. Describe the kernel objects used by a VAXELN system
2. Develop programs in a VAXELN-supported language on a host VMS system
3. Downline load a complete VAXELN system
4. Debug, from a remote host, a VAXELN system

0.2 BEFORE ATTENDING THIS COURSE

Participants should have a sound knowledge of:

1. VMS operating system (preferably V4.0 or later)
2. a language supported by VAXELN

These prerequisites may be achieved by attending a VMS Utilities and Commands course and a Programming in (language) course respectively. Where language is a language supported by VAXELN. Currently these languages are:

- o Ada *
- o C
- o FORTRAN
- o Pascal

*Ada is a registered trademark of the US Government, Ada Joint Program Office

INTRODUCTION TO COURSE

0.3 DOCUMENTATION FOR THE COURSE

Title	Order Number
VAX Language-Sensitive Editor VAXELN Pascal Guide	AA-GR65B-TE
VAXELN C Run-Time Library Reference Manual	AA-EU40B-TE
VAXELN Application Design Guide	AA-EU41B-TE
VAXELN Fortran Programmers Guide	AA-HW72B-TE
VAXELN Host System Guide	AA-JG87A-TE
VAXELN Introduction To VAXELN	AA-JL11A-TE
VAXELN Run-Time Facilities Guide	AA-JM81A-TE
VAXELN Pascal Language Ref. Manual Part 1: Language Elements	AA-JP29A-TE
VAXELN Pascal Language Ref. Manual Part 2: Programming	AA-JN09A-TE
VAXELN Release Notes	AA-Z454F-TE
VAXELN Software Product Description	SPD 28.02.05

These documents are for Software Version 2.3 of VAXELN as of December 1986.

0.4 OBJECTIVES OF THIS STUDENT GUIDE

1. to supplement, but NOT to replace, the documentation set
2. to provide additional examples and reference material
3. to gather useful data together from different sources

0.5 ABBREVIATIONS USED IN THIS HANDOUT

1. ADG - VAXELN Application Design Guide
2. HS - VAXELN Host System Guide
3. INTRO - Introduction to VAXELN
4. LRM - VAXELN Pascal Language Reference Manual
5. RF - VAXELN Run-Time Facilities Guide
6. RTL - VMS Run-Time Library Routines Reference Manual
7. SPD - VAXELN Software Product Description

Abbreviations and their definitions also appear in the index

0.6 CONVENTIONS USED IN THIS HANDOUT

References to documentation appear thus: [HS:5-15] - meaning VAXELN Host System Guide, chapter 5, page 15

This entire student guide was produced using Digital Standard Runoff. The table of contents and index were created by using the RUNOFF/INTERMEDIATE, RUNOFF/CONTENTS and RUNOFF/INDEX commands as appropriate.

CHAPTER 1

VAXELN CAPABILITIES AND APPLICATIONS

1.1 THE NATURE AND PURPOSE OF VAXELN

- a VMS layered product - a toolkit
- for real-time applications like industrial production, robotics, process control
- based on Pascal for easy design and programming. VAXELN Pascal is a superset of ISO Pascal (as defined in ISO/DIS 7185).
- no VMS operating system overheads present - uses the VAXELN kernel for controlling software
- requires VMS host for development work
- provides a multitasking, multiprocessing system
- supports file handling
- network facilities available

1.2 VAXELN SYSTEM COMPONENTS

[SPD:1, INTRO:1-1, RF:1-1]

- the VAXELN kernel controls software and resources
- the software (programs) employed in the system are:
 - (a) DIGITAL supplied
 - (b) user-written

VAXELN CAPABILITIES AND APPLICATIONS

- one or more jobs - VMS definition of a job applies to VAXELN
- the hardware consists of one or more VAXs, peripherals, special interfaces etc.

1.3 FUNCTION OF THE VAXELN KERNEL

[INTRO:3-1, RF: 1-8, SPD:1]

The VAXELN kernel is responsible for:

- manipulating objects called kernel objects
- controlling the sharing of resources
- providing synchronization and communication

The VAXELN kernel comes as a prepared image as part of the software issued when VAXELN is purchased.

1.3.1 User-written Programs

[INTRO:2-1, RF:1-4]

- device drivers
- special applications programs
- number crunchers etc.

1.3.2 DIGITAL Supplied Programs

[INTRO:2-1, RF:1-4]

- servers for file and network handling
- drivers - source code and images supplied
- VAXELN kernel - image form

1.4 A SIMPLE VAXELN APPLICATION

Before starting work on a simple VAXELN application we must satisfy ourselves that we have the minimum hardware and software available to complete and run the application.

1.4.1 Minimum Hardware And Software Requirements

1.4.1.1 Hardware Requirements -

[HS:1-1, SPD:7-8]

- o Development system:

- (a) MicroVAX I
- (b) MicroVAX II
- (c) any 700 series VAX
- (d) any 8000 series VAX

- o Target system:

- (a) MicroVAX I
- (b) MicroVAX II
- (c) VAX-11/725
- (d) VAX-11/730
- (e) VAX-11/750
- (f) VAX 8500
- (g) VAX 8550
- (h) VAX 8700
- (i) KA620

VAXELN CAPABILITIES AND APPLICATIONS

- o Development system memory and space:
 - (a) at least 1 megabyte (Mb) of physical memory
 - (b) 2 Mb of virtual page file quota per user
 - (c) 250-page minimum working set per user
 - (d) 8 000 blocks of disk space for installation
 - (e) 7 500 blocks of disk space for permanent use

- o Target system memory and space:
 - at least 256 kilobytes (Kb) of physical memory made up of components whose sizes are:
 - (a) kernel: 20 Kb
 - (b) language run-time: 128 Kb
 - (c) file service: 50 Kb
 - (d) network service: 24 Kb
 - (e) local debugger: 60 Kb
 - (f) remote debugger: 12 Kb
 - (g) device drivers: 2-7 Kb

- o Target machine must have loading device from:
 - (a) Files-11 disk
 - (b) TU58 cartridge tape
 - (c) TK50 if target is MicroVAX II
 - (d) Ethernet adapter DEQNA (DIGITAL Ethernet to Q-bus network adapter) or DEUNA (DIGITAL Ethernet to Unibus network adapter)

1.4.1.2 Software Requirements -

- o Development system:
 - (a) VMS or MicroVMS operating system
 - (b) DECnet-VAX for downline loading, remote debugging and communications
- o Target system:
 - (a) no software required
 - (b) VAXELN includes target system DECnet licence

1.5 DEVELOPING A SIMPLE APPLICATION

[INTRO:2-3, HS:1-2]

There are six stages in the development of a simple application:

1. edit the program to be run under the VAXELN system
2. compile the program created at 1
3. link the program's object code with the required library routines using LINK
4. build the VAXELN system using EBUILD
5. prepare the target and host for downline loading
6. downline load the system to run the program

VAXELN CAPABILITIES AND APPLICATIONS

1.5.1 Creating A Simple Program

Use a standard VMS text editor to create a source text file. Typical editors that you might use are:

- o EDT
- o EVE (Extensible VAX Editor)
- o LSE (Language Sensitive Editor)

EVE appeared with V4.2 of VMS but LSE is a layered product.

1.5.1.1 Simple Program In Pascal -

```
PROGRAM Simple (OUTPUT);

    (*
        MODULE: SIMPLE.PAS
    *)

BEGIN
    WRITELN ( 'A simple program to test VAXELN' );
END.
```

1.5.1.2 Simple Program In C -

```
main ()
/*
    MODULE:      SIMPLE.C
*/
{
    printf( "A simple program to test VAXELN\n");
}
```

1.5.1.3 Simple Program In FORTRAN -

```
PROGRAM Simple

* MODULE:  SIMPLE.FOR

IMPLICIT NONE
```

```
WRITE (6, '(' A simple program to test VAXELN' )' )
END
```

1.5.1.4 Compiling Our Simple Program -

[HS:2-1]

At your host VAX terminal issue the following command line:

```
$ EPASCAL /DEBUG SIMPLE           ! For Pascal
$ FORTRAN /DEBUG /NOOPTIMIZE SIMPLE ! For FORTRAN
$ CC /DEBUG SIMPLE + ELN$:VAXELNC/LIB ! For C
```

This produces the object file SIMPLE.OBJ containing binary code of our source file plus debug information.

1.5.2 Linking Our Simple Program

[HS:2-8]

At your host VAX terminal issue the following command line:

For Pascal:

```
$ LINK /DEBUG SIMPLE, ELN$:RTLSHARE /LIBRARY, -
_$      RTL /LIBRARY
```

For FORTRAN:

```
$ LINK /DEBUG /NOSYSLIB SIMPLE, ELN$:FRTLOBJECT/LIB, -
_$      RTLSHARE/LIB, RTL/LIB
```

For C:

```
$ LINK /DEBUG SIMPLE, ELN$:CRTLSHARE/LIB, RTL/LIB
```

This command links the object code of SIMPLE.OBJ with the VAXELN run-time library and kernel to produce SIMPLE.EXE.

1.5.3 Building A VAXELN System

[HS:3-1]

At your host VAX terminal issue the following command line:

```
$ EBUILD SIMPLE
```

VAXELN CAPABILITIES AND APPLICATIONS

This reveals a main menu display. Using cursor arrow keys move to "Add Program Description" option. Hit the PF1 key which is the "DO" key. A new menu will appear and the cursor is against the option "Program". Type in the name "SIMPLE" and press PF1. We are returned to the main menu where we return the cursor to "Build System" and press PF1.

The system is being built and, after a few seconds, a message appears giving a full directory specification for SIMPLE.SYS;1 followed by its size in pages and kilobytes.

EBUILD produces two files:

- .SYS - the system image file - binary
- .DAT - menu selections - ASCII

For the simple program we've produced here the output from EBUILD looks something like this:

```
System DISK$INSTRUCT:[SHONE.VAXELN]SIMPLE.SYS;1
```

```
System image size is 285 pages (143K bytes)
```

and the .DAT file contains the single line:

```
program SIMPLE.EXE
```

1.5.4 Preparing For Downline Loading

The Network Control Program (NCP) must be run from your host VAX terminal to check that the target node's details are in the database at the host. Issue the command line:

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

When you receive the NCP> prompt issue the command:

```
NCP> SHOW NODE BEDLAM CHARACTERISTICS
```

Several lines of information may appear. For example:

```
Node Volatile Characteristics as of 30-MAY-1986 13:12:00
```

```
Remote node = 63.736 (BEDLAM)
```

```
Service circuit = UNA-0
```

```
Hardware address = AA-00-03-01-34-59
```

```
Load file = SYS$SYSROOT:[SYSMGR]SIMPLE.SYS
```

If no information is forthcoming please follow the instructions laid down at

[HS:4-4].

1.5.5 Downline Loading A Simple System

This involves informing the host VAX DECnet database of the file to send down the Ethernet to the target machine.

For example issue the command:

```
NCP> SET NODE BEDLAM LOAD FILE DISK$INSTRUCT:[SHONE.VAXELN]SIMPLE.SYS
```

Next go to the target MicroVAX and place it in console mode. The procedure is different for each type of MicroVAX, as follows:

- MicroVAX I - depress the HALT button on the front panel TWICE
- MicroVAX II - first check that the HALT enable button on the KA 630 panel (at rear) is set to enable (dot in circle) then proceed as for MicroVAX I

This halt operation produces a prompt thus >>> on the MicroVAX console terminal. Enter the command B XQA0 from the prompt and after a minute or two from pressing <RET> you should see 'A simple program to test VAXELN' on the screen - your simple program's output.

If you would like to downline load the same program using EDEBUG [HS:5-4] onwards gives details. We shall be covering this alternative loading method in more detail later in the course.

CHAPTER 2

VAXELN EXTENSIONS TO ISO PASCAL

2.1 SOURCE TEXT

[LRM:1-7, Program INCLUDE_SOURCE.PAS]

External source text may be included in a compilation using the construction:

```
        %INCLUDE 'FILE_OPEN';  
or      %INCLUDE 'FILE_OPEN/LIST';  
or      %INCLUDE 'FILE_OPEN/NOLIST';
```

where FILE_OPEN is of type .PAS

Whether listing is performed is decided ultimately by command qualifiers to EPASCAL.

2.1.1 Identifiers In VAXELN Pascal

[LRM:1-2]

- Have a maximum length of 31
- Allowed characters are:
 - (a) 0 1 2 3 4 5 6 7 8 9
 - (b) A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 - (c) a b c d e f g h i j k l m n o p q r s t u v w x y z
 - (d) \$ and _

VAXELN EXTENSIONS TO ISO PASCAL

- the dollar character is best avoided in user-defined identifiers because of possible clashes with DIGITAL-defined symbols.

2.1.2 Reserved Words In VAXELN Pascal

[LRM:1-2]

The following are additional reserved words in VAXELN Pascal:

- FUNCTION_BODY
- INTERRUPT_SERVICE
- MODULE
- OTHERWISE
- PROCEDURE_BODY
- PROCESS_BLOCK

2.1.3 Program Structure

[LRM:2-1, Program MOD-PROG-PROC-1.PAS]

In VAXELN Pascal programs are executed as jobs. A job is a master process and zero or more subprocesses. The principal code segment of a job is called a program block. Within a program block other routines may be invoked, for example:

- standard Pascal FUNCTIONS
- standard Pascal PROCEDURES
- VAXELN Pascal process blocks
- VAXELN Pascal interrupt service routines

A process block is invoked using the procedure `CREATE_PROCESS`. Interrupt service routines are invoked asynchronously when a device interrupt occurs to which the routine has been attached by a call to the procedure `CREATE_DEVICE`.

2.1.4 VAXELN MODULE

[LRM:2-6]

A VAXELN MODULE comprises:

- a set of outer-level declarations
- may contain module headers more explicitly specifying:
 - (a) names to be EXPORTed by the module
 - (b) names exported by others to it (IMPORTed)
 - (c) names of other modules to be used in the compilation

An example of the relationship between modules, programs and other routines appears at [LRM:2-3].

2.1.5 Module Headers

[LRM:2-7]

- they may be preceded by comments
- may be followed by:
 - (a) EXPORT - the default for outer-level declarations
 - (b) IMPORT
 - (c) INCLUDE
- may use attributes:
 - (a) GLOBALDEF
 - (b) IDENT (string)

VAXELN EXTENSIONS TO ISO PASCAL

2.1.6 Export Headers

[LRM:2-9, Program IMPORT_1.PAS]

- list names to be exported
- names must be declared at outer level
- absence of EXPORT implies export of ALL outer-level declarations
- an empty EXPORT header is valid - may be used for separate routine body
- GLOBALDEF attribute implies all ordinal constants exported are available to VMS linker as global values

2.1.7 Import Headers

[LRM:2-10, Program IMPORT_1.PAS]

- list names to be imported into compilation of module
- names are exported from another module

2.1.8 Include Headers

[LRM:2-10, Program INCLUDE_1.PAS]

- lists object modules to be included in compilation
- may be affected by EPASCAL command qualifier

2.1.9 Program Block

[LRM:2-12]

- the reserved word PROGRAM may be preceded by one of the attributes:
 - (a) UNDERFLOW
 - (b) NOUNDERFLOW
- only one PROGRAM block declaration is allowed in a complete VAXELN Pascal program
- program may have string arguments of up to 100 characters per argument
- arguments are specified either:
 - (a) when invoking CREATE_JOB or
 - (b) at system build time
- Two functions are provided to permit access to program arguments:
 - (a) PROGRAM_ARGUMENT - to obtain value of argument
 - (b) PROGRAM_ARGUMENT_COUNT - to obtain number of arguments
- when program execution completes a termination message may be sent to a specified port if job containing it was created by CREATE_JOB with the NOTIFY parameter

2.2 DATA TYPES

[LRM: 3-1]

2.2.1 Ordinal Types

Internal representation of ordinal types may be controlled by preceding their declaration by one of the size attributes:

VAXELN EXTENSIONS TO ISO PASCAL

- BIT
- BYTE
- WORD
- LONG

2.2.2 Enumerated Types

[LRM:3-9]

- up to 32 767 named values

2.2.3 Set Types

[LRM:3-12]

- base type MUST be ordinal
- set of integer limited to values in range 0-32 766

2.2.4 Flexible Types

[LRM:3-17]

Flexible types are types with parameters that specify lengths or extents. The three predeclared flexible types are:

- STRING
- VARYING_STRING
- BYTE_DATA

A flexible type is limited to one of four type definitions as follows:

- pointer to a bound flexible type
- another flexible type
- a record type
- an array type

2.2.5 Bound Flexible Type

[LRM:3-19]

To make use of a flexible type you must do the following to provide a bound flexible type:

- specify a type name
- specify values for the extent parameters

For example:

```
TYPE      Matrix (x,y:1..10) = ARRAY [1..x,1..y] OF REAL;
VAR       Mat_A : Matrix(5,8);
```

Extents may be specified as ordinal-valued expressions in bound flexible type definitions.

Extent expressions may also be used in the declaration of:

- CONST
- VAR
- upper and lower bounds of arrays

2.2.6 String Types

[LRM:3-23]

- sequence of 0 to 32 767 characters

VAXELN EXTENSIONS TO ISO PASCAL

- data types are:

- (a) STRING
- (b) VARYING_STRING

(In addition there is the standard Pascal PACKED ARRAY OF CHAR)

!

2.2.7 Array Types

[LRM:3-26]

- total number of dimensions not to exceed 8
- may be qualified by ALIGNED attribute

2.2.8 Record Types

[LRM:3-32, Program ATTRIBS_1.PAS]

- POS attribute is valid on field of packed record
- ALIGNED attribute is valid on a record
- ALIGNED attribute is valid on a field of a record provided that POS is not present there

2.2.9 Pointer Types

[LRM:3-39]

- allowed to point to data type ANYTYPE
- ANYTYPE is completely unspecified
- ^ANYTYPE implies that references to data must use typecasting

2.2.10 System Data Types

[LRM:3-45]

- AREA - 32 bits
- DEVICE - 32 bits
- EVENT - 32 bits
- MESSAGE - 32 bits
- NAME - 32 bits
- PORT - 128 bits
- PROCESS - 32 bits
- SEMAPHORE - 32 bits

2.2.11 Miscellaneous Predeclared Data Types

[LRM:3-53]

- BYTE_DATA - a number of 8-bit bytes
- LARGE_INTEGER - 64-bit integer

2.2.11.1 BYTE_DATA Data Type -

[LRM:3-53, Programs ATTRIBS_1.PAS, TYPECAST_1.PAS]

- no predefined operations on this type except:
 - (a) assignment
 - (b) argument passing
- parameters of BYTE_DATA(n) are taken as compatible with data of any size

VAXELN EXTENSIONS TO ISO PASCAL

- conformant `BYTE_DATA(<n>)` are compatible with data of any size
- size of `BYTE_DATA` may be omitted in typecast operations

2.2.11.2 `LARGE_INTEGER` Data Type -

[LRM:3-54, Program `TIME_1.PAS` and others in Appendix D]

- signed integers but not ordinal
- useful for time values
- bit 63 is significant i.e. clear for positive and set for negative values

2.3 ATTRIBUTES FOR DATA SIZING

[LRM:3-63, Appendix A]

- `BIT (extent expression)` [LRM:3-64, Program `ATTRIBS_1.PAS`]
- `BYTE` [LRM:3-65, Program `TYTPECAST_1.PAS`]
- `WORD` [LRM:3-65, Program `ATTRIBS_1.PAS`]
- `LONG` [LRM:3-65]
- `ALIGNED (extent expression)` [LRM:3-65]

In using the `BIT` attribute the extent expression must produce an integer constant in the range 1 to 32

`ALIGNED` is valid on:

- array type definition
- record type definition
- field in a packed record definition provided `POS` has not been used on it

The extent expression used with `ALIGNED` must render an integer constant in the range 0 to 2. These values have the following meanings:

- 0 - aligned on BYTE boundary
- 1 - aligned on WORD boundary
- 2 - aligned on LONG boundary

2.4 CONSTANTS

[LRM: 4-1]

The virtues of using constants wherever and whenever possible are just as important in VAXELN as they are in any programming.

2.4.1 Non-decimal Radix Specifiers

[LRM: 4-3, Program RADIX_1.PAS]

From time to time it is convenient to specify a constant e.g. system symbol, in its original radix e.g. hexadecimal. VAXELN Pascal allows this specification

2.4.1.1 Radix Specifiers: -

- %b or %B - binary (base 2)
- %o or %O - octal (base 8)
- %x or %X - hexadecimal (base 16)

The specifier is followed by an unsigned series of digits with optional apostrophes enclosing them e.g.:

```
%b'10101010'
%B 01010101
%b '00010111 11101111 00000011 11100000'
```

The form that uses apostrophes allows embedded spaces and tabs permitting formatting for improved legibility and clarity.

VAXELN EXTENSIONS TO ISO PASCAL

Note that the characters following a radix specifier must be valid for that radix:

- binary - 1, 0
- octal - 0, 1, 2, 3, 4, 5, 6, 7
- hexadecimal - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, a, B, b, C, c, D, d, E, e, F, f

2.4.2 Non-printing Characters In Constants

[LRM:4-6, Module INCLUDE_1_DEFS.PAS]

To introduce non-printing character(s) into a string constant enclose ASCII collating sequence value of character (returned by ORD function) in parentheses.

2.5 INITIALIZERS

[LRM:4-8]

o provide constant initial values for:

- (a) variables in VAR section
- (b) defaults for optional value parameters

For example:

```
VAR
    Year:          1900..2099 := 1986;
    Software:      (VMS,RSX,RSTS,VAXELN) := VAXELN;
    Pi:            REAL := 3.14159;
    Machines:      SET OF (VAX, MicroVAX, PDP) := [VAX];

PROCEDURE Ring_bell (Rings : 1..15 := 1);
```

2.5.1 Initializing To Binary Zero

[LRM:4-10]

VAXELN Pascal provides the function ZERO. It is compatible with any data type.

For example

```
VAR
  Counter    : INTEGER := ZERO;
  Average    : REAL     := ZERO;
  RMS_value  : DOUBLE   := ZERO;
```

2.5.2 Initializing Arrays And Records (Aggregate Initialization)

[LRM:4-11, Program AGGREG_1.PAS]

Example

```
VAR
  Matrix      : ARRAY [1..3, 1..6, 1..9] OF 1..15 :=
                (3 OF (6 OF (9 OF 2)));

  Part_rec   : RECORD
    Marker    : BIT(1) 0..1;
    Total     : ALIGNED(2) REAL
  END := (1, 0.0);
```

Care should be exercised in using any form of initializer for reasons detailed at [LRM:4-12].

2.6 PREDECLARED NAMED CONSTANTS

[LRM:4-13]

Standard Pascal TRUE, FALSE and MAXINT plus:

- o ASSERT_CHECK_ENABLED - has values TRUE or FALSE

This constant is used to make execution of code dependent upon the presence of the EPASCAL qualifier CHECK=ASSERT

VAXELN EXTENSIONS TO ISO PASCAL

2.7 PREDECLARED ENUMERATED TYPES

[LRM:4-14]

Enumerated Type	Values
EVENT_STATE	EVENT\$CLEARED, EVENT\$SIGNALLED
FILE_ACCESS	ACCESS\$SEQUENTIAL, ACCESS\$DIRECT
FILE_CARRIAGE_CONTROL	CARRIAGE\$LIST, CARRIAGE\$FORTRAN, CARRIAGE\$NONE
FILE_DISPOSITION	DISPOSITION\$SAVE, DISPOSITION\$DELETE
FILE_HISTORY	HISTORY\$OLD, HISTORY\$NEW, HISTORY\$UNKNOWN, HISTORY\$READONLY
FILE_RECORD_TYPE	RECORD\$FIXED, RECORD\$VARIABLE
FILE_SHARING	SHARE\$NONE, SHARE\$READONLY, SHARE\$READWRITE
NAME_TABLE	NAME\$LOCAL, NAME\$UNIVERSAL, NAME\$BOTH
OPEN_CIRCUIT	CIRCUIT\$CONNECT, CIRCUIT\$ACCEPT
QUEUE_POSITION	QUEUE\$HEAD, QUEUE\$TAIL, QUEUE\$CURRENT

2.8 VARIABLES

[LRM:5-1]

Attributes may be applied to the declaration of variables. In particular the following may appear:

- o ALIGNED
- o BIT
- o BYTE
- o EXTERNAL
- o LONG
- o READONLY
- o VALUE
- o WORD

The use of ALIGNED, BIT, BYTE, LONG and WORD has been described elsewhere

2.8.1 READONLY Attribute

[LRM:5-3, Appendix A]

- o variable allocated in readonly storage
- o variable must have an initializer unless EXTERNAL attribute also applies
- o variable must not be a file or contain a file

2.8.2 VALUE Attribute

[LRM:5-4, Appendix A]

- o provides information to the VMS linker about data item

VAXELN EXTENSIONS TO ISO PASCAL

- o without EXTERNAL, name is available to linker as value of a global symbol
- o with EXTERNAL, name is value to be supplied to linker by non-VAXELN Pascal module
- o without EXTERNAL, declaration must have initializer
- o variable reference to item must be its name only - typecasting it is forbidden
- o data type must be represented in less than or equal to 32 bits
- o data type must not have BIT attribute

2.8.3 EXTERNAL Attribute

[LRM:5-5; Appendix A, Program TIME_4.PAS]

- o variable is defined in non-VAXELN Pascal module
- o with additional VALUE attribute, item must be available to the VMS linker as a global value
- o without VALUE attribute must be available as global symbol from another module

2.8.4 Pseudo Variable References

[LRM:5-8]

Pseudo variable references may be formed when using the predeclared functions SUBSTR [LRM:9-22] or ARGUMENT [LRM:9-42]:

- o invoking SUBSTR with a variable reference as its first argument (string)
- o invoking ARGUMENT with a VAR parameter as its first argument (parameter_name)

2.8.5 Typecast Variable References

[LRM:5-10, Programs TYPECAST_1.PAS, COMM_9.PAS]

Typecasting allows a programmer to relax, for one assignment, the strict rules of Pascal regarding type compatibility. For example:

```
VAR
    A : REAL;
    B : INTEGER;
.
.
.
    A :: INTEGER := B;
```

The double colon is the operator and there must not be a space between the two colons. The named type - to the right of the operator - may be:

- a type name
- a bound flexible type
- a pointer to a type name
- a pointer to a bound flexible type

Typecasting to BYTE_DATA does not require a storage size specification, for example:

```
VAR
    A : REAL;    B : INTEGER;
.
.
    A :: BYTE_DATA := B :: BYTE_DATA;
```

Typecasting to a flexible type allows arbitrary extent expressions not just the special extent expressions. The expressions are evaluated each time the typecast reference occurs. For example:

```
TYPE
    List_type (n:INTEGER) = ARRAY[1..n] OF REAL;

VAR
    Data_list : List_type(20);
    Dummy : INTEGER;
    I      : INTEGER := 10;
.
.
    Dummy := Data_list [ I*5/10 ] :: INTEGER;
```

VAXELN EXTENSIONS TO ISO PASCAL

Typecasting is not allowed on:

- o literals
- o named constants

2.8.6 Addresses Of Variables

[LRM:5-13]

Normally the virtual address of a variable may be found using the predeclared function ADDRESS. However the ADDRESS function may not be used in the following circumstances:

- o data items not on a byte boundary
- o when data type has bit alignment AND reference has a bit offset from an addressable location

2.9 ALLOCATION OF STORAGE TO CONSTANTS AND VARIABLES

Details will be found on pages 18 and 19 of Chapter 5 in the VAXELN Pascal Language Manual. It is worth noting that the data PSECT in a program is shared by all processes in a job. Separate data PSECTs are created for new jobs running a program that requires non-zero initializers.

2.9.1 Sharing Data Between Processes

[LRM:5-15]

Data may be shared by one or more processes in a job. The exceptions are:

- o local variables of routines
- o value parameters

These items go into process private P1 virtual memory on the per-process stack. Sharing of outer-level data may be achieved by:

- o referencing a variable by name

- o using pointers
- o VAR parameters of process blocks

2.9.2 Shared Data - Care In Modifying

[LRM:5-15]

Several routines exist that perform atomic operations and can be used safely on data shared by processes within a job:

- o READ_REGISTER - predeclared procedure
- o WRITE_REGISTER - predeclared procedure
- o INSERT_ENTRY - predeclared procedure
- o REMOVE_ENTRY - predeclared procedure
- o ADD_INTERLOCKED - predeclared function

2.9.3 READ_REGISTER, WRITE_REGISTER

[LRM:5-15; 14-36 to 14-40]

These predeclared procedures, though intended for operations involving device registers, provide a safe method for accessing and/or modifying shared variables. The operation, in each case, is performed by a single appropriate VAX instruction of the MOVE type.

2.9.4 INSERT_ENTRY, REMOVE_ENTRY

[LRM:5-15; 10-5 to 10-9]

These predeclared procedures allow insertion and removal of entries from the head or tail of a queue. The underlying single VAX instruction, INSQUE and REMQUE respectively, is a non-interruptible instruction.

VAXELN EXTENSIONS TO ISO PASCAL

2.9.5 ADD_INTERLOCKED

[LRM:5-20; 9-69]

This predeclared function allows addition of an integer in the range -32768 to 32767 to a target WORD integer (16 bits) by using a single VAX instruction, Add Aligned Word Interlocked (ADAWI). This is a non-interruptible instruction and will not permit multiple simultaneous access to a shared variable.

2.10 INITIALIZATION OF SHARED DATA

[LRM:5-16]

The recommended practice for shared outer-level variables is to initialize them from the master process BEFORE subprocesses are created. This is a simple but effective method of synchronization.

2.11 EXPRESSIONS AND OPERATORS

[LRM:6-1]

The additive dyadic and monadic operators are valid on LARGE_INTEGERS. Further, LARGE_INTEGERS may be combined with INTEGERS.

2.11.1 Exponentiation Operator

[LRM:6-11]

VAXELN Pascal includes the exponentiation operator ** amongst its set of arithmetic operators. The operands are restricted as follows:

- o first operand - REAL or DOUBLE
- o second operand - REAL, DOUBLE or INTEGER

2.11.2 Mixed Operands

[LRM:6-9]

The monadic operators + and - do not affect the result of their operation so far as data type is concerned. The dyadic operators affect the result type as follows:

First operand	Second operand			
	INTEGER	LARGE_INTEGER	REAL	DOUBLE
INTEGER	INTEGER	LARGE_INTEGER	REAL	DOUBLE
LARGE_INTEGER	LARGE_INTEGER	LARGE_INTEGER	N/V	N/V
REAL	REAL	N/V	REAL	DOUBLE
DOUBLE	DOUBLE	N/V	DOUBLE	DOUBLE

N/V = not valid

2.12 CONCATENATING STRINGS

[LRM:6-18, Program TIME_3.PAS]

To concatenate two string expressions use the dyadic addition operator '+'

2.13 STATEMENTS IN VAXELN PASCAL

[LRM: 7-1]

2.13.1 Labels

[LRM:7-2]

Labels in VAXELN Pascal may be either:

- o literal integer constants - as ISO Pascal or
- o a valid identifier

Explicit declaration of labels is not required as their use declares them implicitly. However the LABEL declaration remains valid to conform, and remain compatible, with ISO Pascal. This is recommended as good programming style.

VAXELN EXTENSIONS TO ISO PASCAL

2.13.2 CASE Statement

[LRM:7-9]

The standard Pascal CASE statement is extended by the addition of the OTHERWISE clause. This allows some action to be defined if none of the cases is selected. Without the OTHERWISE clause, failure to satisfy one of the cases causes a range violation at run time. The range of values in the case constants must not exceed 32 767.

2.14 PROCEDURES AND FUNCTIONS

[LRM:8-1]

There are a number of extensions to ISO Pascal. [LRM:8-1 ->] is definitive.

2.14.1 Declaring Procedures And Functions

[LRM:8-2]

Headings may include the following directives:

- o FUNCTION_TYPE [Program LAB_1.PAS]
- o PROCEDURE_TYPE

thus declaring a particular type of function or procedure:

```
FUNCTION Temperature (Old_temp : REAL) : REAL; FUNCTION_TYPE;

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature;

BEGIN .....
```

Other directives that may be used are:

- o EXTERNAL
- o FORWARD - standard Pascal
- o SEPARATE

Separate and FORWARD inform the EPASCAL compiler that the text of the routine's body is defined elsewhere while EXTERNAL indicates that the routine's body is defined in another programming language

2.15 QUEUES

[LRM:10-1]

Queues are efficiently handled in VAXELN Pascal using the predeclared data type `QUEUE_ENTRY` and the interface to the VAX instructions `INSQUE` and `REMQUE` provided by the procedures `INSERT_ENTRY` and `REMOVE_ENTRY`. Queues are started using the procedure `START_QUEUE`. The definition of `QUEUE_ENTRY` provides a forward and backward link just as in the queues manipulated by the VMS operating system.

CHAPTER 3

KERNEL OBJECTS AND THEIR USE

3.1 WHAT ARE KERNEL OBJECTS?

[INTRO:3-1, RF:2-1]

The kernel objects are data structures acted upon by the kernel and which represent resources, processes etc. These objects are protected and are inaccessible from programs except via special procedures. When one of these procedures is invoked to create an object the kernel allocates, dynamically, a block of memory for the object and returns an identifying value for it. This value is useful for program references to an object as well as in the deletion of an object.

The VAXELN kernel objects are:

- o AREA
- o DEVICE
- o EVENT
- o MESSAGE
- o NAME
- o PORT
- o PROCESS
- o SEMAPHORE

KERNEL OBJECTS AND THEIR USE

3.1.1.1 AREA Object

[INTRO:3-2, RF:2-4, Program COMM_5.PAS]

- represents a region of physical memory
- the region may be shared among jobs on a single node
- contains a binary semaphore to synchronize access
- may have a size of 0 representing just the binary semaphore
- has a name of up to 31 characters
- has state SIGNALED or FREE
- has list of processes waiting for access to the region
- has region attached to it
- the object itself occupies one block of kernel pool (128 bytes)
- the region is mapped into process P0 space
- region is allocated from physically contiguous 512-byte pages of memory

VAXELN AREAs are similar to the VMS shared regions and global sections

3.1.1.1.1 Operations on AREA objects

The following predeclared routines may be used to manipulate AREA objects:

- CREATE_AREA - creates, or maps existing area; returns ID of area and a pointer to region of memory
- WAIT_ALL, WAIT_ANY - for gaining exclusive access to an area process waits for the signalling of an area by passing the area value to one of these procedures
- SIGNAL - an area is signalled by passing its value to this predeclared procedure
- DELETE - an area is deleted from an application by passing its value to this procedure

3.1.1.2 Call format for CREATE_AREA

The call format for CREATE_AREA is:

```
CREATE_AREA (  area,
               data_pointer,
               area_name,
               VIRTUAL := base_va,
               STATUS  := stat
            );
```

3.1.2 DEVICE Object

[INTRO:3-3, RF:2-11]

- enables interrupt service routine (ISR) to signal interrupt to process
- ISR called by kernel when interrupt occurs
- signalling a device object enables synchronization with processes in job
- device object has set of device characteristics established with system builder
- has a communication region
- ISR is passed DEVICE value and communication region on interrupt
- DEVICE object occupies one block of kernel pool (128 bytes)
- connected ISR requires one block of kernel pool for its dispatcher

3.1.2.1 Operations on DEVICE objects

The following predeclared routines may be used to manipulate DEVICE objects:

- CREATE_DEVICE - creates a DEVICE object and returns its ID
- WAIT_ALL, WAIT_ANY - processes wait for the signalling of a DEVICE object from an Interrupt Service Routine (ISR) by passing the DEVICE value to one of these procedures

KERNEL OBJECTS AND THEIR USE

- `SIGNAL_DEVICE` - a `DEVICE` is signalled from an (ISR) by passing its value to this predeclared procedure
- `DELETE` - a `DEVICE` is deleted from an application by passing its value to this procedure

3.1.2.2 Call format for `CREATE_DEVICE`

The call format for `CREATE_DEVICE` is:

```
CREATE_DEVICE ( device_name,  
                device,  
                VECTOR_NUMBER    := relative_vector,  
                SERVICE_ROUTINE  := routine_name,  
                REGION           := region_pointer,  
                REGISTERS        := register_pointer,  
                ADAPTER_REGISTERS := adapter_pointer,  
                VECTOR           := vector_pointer,  
                PRIORITY         := interrupt_priority,  
                POWERFAIL_ROUTINE := power_routine,  
                STATUS           := stat  
            );
```

3.1.3 EVENT Object

[INTRO:3-2, RF:2-5, Program SYNCH_2.PAS and others]

- records occurrences of events until cleared
- state `SIGNALED` or `CLEAR`
- has list of processes waiting for event to be signaled
- `EVENT` objects occupy one block of kernel pool (128 bytes)

`VAXELN` `EVENT` objects are very similar to the event flags of `VMS`

3.1.3.1 Operations On `EVENT` Objects -

The following predeclared routines may be used to manipulate `EVENT` objects:

- `CREATE_EVENT` - creates an `EVENT` object and returns its ID
- `WAIT_ALL`, `WAIT_ANY` - processes wait for the signalling of an `EVENT` object by passing the `EVENT` value to one of these procedures
- `SIGNAL` - an `EVENT` is signalled by passing its value to this predeclared procedure
- `CLEAR_EVENT` - an event is cleared by passing its value to this procedure
- `DELETE` - an event is deleted from an application by passing its value to this procedure

3.1.3.2 Call format for `CREATE_EVENT`

The call format for `CREATE_EVENT` is:

```
CREATE_EVENT (  event,
                initial_state,
                STATUS := stat
                );
```

KERNEL OBJECTS AND THEIR USE

3.1.4 MESSAGE Object

[INTRO:3-2, RF:2-7, Program COMM_1.PAS and others]

- used when sending data from a job to a PORT
- PORT usually in another job
- MESSAGE contains data and its length
- data are mapped in process P0 space
- creation returns identifier and pointer to data
- MESSAGE sent by providing MESSAGE and PORT values to SEND procedure which removes message from sender's P0 space
- MESSAGE removed from PORT and mapped into P0 space by providing PORT value to RECEIVE procedure
- RECEIVE returns identifier and pointer to message data
- MESSAGE object occupies one block of kernel pool (128 bytes)
- message data are allocated from physically contiguous 512-byte pages of memory, page aligned

3.1.4.1 Operations On MESSAGE Objects -

In addition to the SEND and RECEIVE procedures noted above the following predeclared routines may be used to manipulate MESSAGE objects:

- CREATE_MESSAGE - creates a MESSAGE object returns its ID and maps its data into the job's P0 address space
- DELETE - a MESSAGE is deleted from an application by passing its value to this procedure

3.1.4.2 Call format for CREATE_MESSAGE

The call format for CREATE_MESSAGE is:

```
CREATE_MESSAGE ( message,  
                data_pointer,  
                STATUS := stat );
```

3.1.5 NAME Object

[INTRO:3-3, RF:2-10, Program COMM_3.PAS and others]

- entry in a name table
- associates a character string with a message port
- there are two types of name:
 - (a) local names - within a node
 - (b) universal names - at all nodes
- universal name requires 64 bytes of kernel pool in local network service and 64 bytes in network service of network's current name server.
- name may be up to 31 characters
- has the PORT value identifying the object
- NAME object occupies one block of kernel pool (128 bytes)

3.1.5.1 Operations On NAME Objects -

The following predeclared routines may be used to manipulate NAME objects:

- CREATE_NAME - creates a NAME and returns its value
- TRANSLATE_NAME - provides an associated PORT value from the name string supplied to this procedure
- DELETE - a NAME is deleted from an application by passing its value to this procedure

3.1.5.2 Call format for CREATE_NAME

The call format for CREATE_NAME is:

```
CREATE_NAME    ( name,
                name_string,
                port_value,
```

KERNEL OBJECTS AND THEIR USE

```
TABLE    := table,  
STATUS   := stat  
);
```

3.1.6 PORT Object

[INTRO:3-3, RF:2-8, Program COMM_7.PAS and others]

- destination for messages
- each port belongs to a job
- they are accessible from any job in local area network (LAN)
- identifying value is valid in all jobs in all nodes in network
- each executing job in the system has a job port
- ports have maximum number of queued messages
- they have a list of queued messages - removed by RECEIVE procedure
- have state of CONNECTED or UNCONNECTED
- if it is connected, the PORT value of the PORT to which it is connected
- PORT value is 128 bits long [RF:2-9]
- PORT object occupies one block of kernel pool (128 bytes)

3.1.6.1 Operations On PORT Objects -

The following predeclared routines may be used to manipulate PORT objects:

- CREATE_PORT - creates a PORT object and returns its ID
- JOB_PORT - this procedure enables each job to obtain its unique port value
- ACCEPT_CIRCUIT - provides a wait mechanism for a process. A process can wait for a circuit connection request using this procedure
- CONNECT_CIRCUIT - connects a port in a circuit

- DISCONNECT_CIRCUIT - disconnects a port from a circuit
- WAIT_ALL, WAIT_ANY - processes wait for the receipt of a message by passing the PORT value to one of these procedures
- DELETE - a PORT is deleted from an application by passing its value to this procedure

3.1.6.2 Call format for CREATE_PORT

The call format for CREATE_PORT is:

```
CREATE_PORT    ( port,
                LIMIT    := int,
                STATUS    := stat
                );
```

3.1.7 PROCESS Object

[INTRO:3-2, RF:2-3, Program COMM_1.PAS and others]

- represents current execution context in a program in a job
- job is defined as for VMS i.e. a set of cooperating processes
- has one of 16 levels of process priority
- has state of: running, ready, waiting or suspended
- has username and user identification code (UIC)

3.1.7.1 Operations On PROCESS Objects -

The following predeclared routines may be used to manipulate PROCESS objects:

- CREATE_PROCESS - creates a PROCESS and returns its ID
- CURRENT_PROCESS - this procedure enables a process to obtain its own value

KERNEL OBJECTS AND THEIR USE

- SUSPEND - allows suspension of a process's execution
- RESUME - allows resumption of a suspended process
- SET_PROCESS_PRIORITY - allows alteration of a process's priority
- WAIT_ALL, WAIT_ANY - a process waits for another to finish by passing the process value to one of these procedures
- SIGNAL - a process is forced into an exception condition by passing the process value to this procedure
- EXIT - allows a forced immediate exit from a process
- DELETE - a process is deleted from an application by passing its value to this procedure

3.1.7.2 Call format for CREATE_PROCESS

The call format for CREATE_PROCESS is:

```
CREATE_PROCESS    ( process,
                   subprocess_name,
                   argument_list,
                   EXIT      := exit_status,
                   STATUS    := stat
                   );
```

3.1.8 SEMAPHORE Object

[INTRO:3-2, RF:2-6]

- controls and protects resource from simultaneous accessors
- maintains count of number of processes that may be allowed to obtain semaphore
- maximum value for count - maximum number of processes that may have semaphore simultaneously
- list of processes awaiting signalling of semaphore
- SEMAPHORE object occupies one block of kernel pool (128 bytes)

3.1.8.1 Operations On SEMAPHORE Objects -

The following predeclared routines may be used to manipulate SEMAPHORE objects:

- CREATE_SEMAPHORE - creates a SEMAPHORE and returns its ID
- WAIT_ALL, WAIT_ANY - a process waits for the signalling of a semaphore by passing the process value to one of these procedures. The semaphore count is decremented on satisfaction of the wait.
- SIGNAL - a semaphore is signalled by passing its value to this procedure
- DELETE - a semaphore is deleted from an application by passing its value to this procedure

3.1.8.2 Call format for CREATE_SEMAPHORE

The call format for CREATE_SEMAPHORE is:

```
CREATE_SEMAPHORE    ( semaphore,
                    initial_count,
                    maximum_count,
                    STATUS := stat
                    );
```


CHAPTER 4

PROGRAM DEVELOPMENT

4.1 COMPILING VAXELN SOURCES

[HS:2-1, LRM:16-1]

The compilation command is the same as for all VMS native mode high level languages:

```
$ EPASCAL qualifier-list file-specification-list
```

The default file type is .PAS and the result of the compilation is one object file (type .OBJ). The default qualifiers in effect render the EPASCAL command:

```
$ EPASCAL /NOCHECK /NOCROSS_REFERENCE /DEBUG=TRACEBACK -  
_ $ /EXPORT /NOG_FLOATING /INLINE /NOLIST /NOMACHINE_CODE -  
_ $ /NOMAP /OBJECT /OPTIMIZE /SHOW=(SOURCE,HEADER) -  
_ $ /VALIDATE=REQUIRED /WARNINGS
```

Chapter 16 of the VAXELN Pascal Language Reference Manual (Part 2) is definitive on qualifiers. If a particular set of qualifiers is required routinely then the EPASCAL command might be redefined in a LOGIN.COM file. For example:

```
$ !      Redefinition of EPASCAL for normal use  
$ EP*ASCAL == "EPASCAL /LIST /CROSS_REFERENCE /CHECK=ALL"  
$ !  
$ !      Redefinition of EPASCAL for using DEBUG  
$ EPD      == "EPASCAL /DEBUG /LIST /CROSS_REFERENCE /CHECK=ALL"
```

PROGRAM DEVELOPMENT

The full list of qualifiers to EPASCAL is:

<u>Qualifier</u>	<u>Negative form</u>	<u>Default</u>
CHECK=(list) list: ALL ASSERT, NOASSERT, RANGE, NORANGE	NOCHECK	NOCHECK
CROSS_REFERENCE	NOCROSS_REFERENCE	NOCROSS_REFERENCE
DEBUG=(list) list: ALL EXPORT_ONLY IMPORT_TOO NONE SYMBOLS TRACEBACK	NODEBUG	DEBUG=TRACEBACK
EXPORT	NOEXPORT	EXPORT
G_FLOATING	NOG_FLOATING	NOG_FLOATING
INCLUDE=(module-list)	-	-
INLINE	NOINLINE	INLINE
LIBRARY	-	-
LIST=file-specification	NOLIST	NOLIST
MACHINE_CODE	NOMACHINE_CODE	NOMACHINE_CODE
MAP=option option: LOCAL REFERENCED ALL	NOMAP	NOMAP
MODULE	-	-
OBJECT=file-specification	NOOBJECT	OBJECT=source-name.OBJ

<u>Qualifier</u>	<u>Negative form</u>	<u>Default</u>
OPTIMIZE=(option-list) option-list: COMMON_SUBEXPRESSIONS, NOCOMMON_SUBEXPRESSIONS DISJOINT, NODISJOINT INVARIANT, NOINVARIANT LOCALS_IN_REGISTERS, NOLOCALS_IN_REGISTERS PEEPHOLE, NOPEEPHOLE RESULT_INCORPORATION, NORESULT_INCORPORATION	NOOPTIMIZE	OPTIMIZE= (COMMON_SUBEXPRESSIONS, DISJOINT, INVARIANT, LOCALS_IN_REGISTERS, PEEPHOLE, RESULT_INCORPORATION)
SHOW=(option-list) option-list: HEADER, NOHEADER INCLUDE, NOINCLUDE MODULES, NOMODULES SOURCE, NOSOURCE STATISTICS, NOSTATISTICS	-	SHOW=(SOURCE,HEADER)
VALIDATE=option option: NONE REQUIRED ALL	-	VALIDATE=REQUIRED
WARNINGS	NOWARNINGS	WARNINGS

PROGRAM DEVELOPMENT

4.2 USING THE DCL COMMAND LINK

[HS:2-8]

The link command is the same as for all VMS native mode high- level languages:

```
$ LINK qualifier-list file-specification-list
```

4.2.1 VAXELN Object Libraries

The link command requires the presence of two libraries for most linker operations involving VAXELN Pascal images. These libraries are:

- o ELN\$:RTLSHARE.OLB
- o ELN\$:RTL.OLB

It may be convenient to take advantage of the VMS LNK\$ logical names thus:

```
$ ASSIGN ELN$:RTLSHARE.OLB LNK$LIBRARY
$ ASSIGN ELN$:RTL.OLB      LNK$LIBRARY_1
```

If these assignments are placed in a LOGIN.COM file or established during a VAXELN work session DCL command LINK may be reduced from:

```
$ LINK files-to-be-linked ,ELN$:RTLSHARE/LIBRARY -
_$                        ,ELN$:RTL/LIBRARY
```

```
$ LINK files-to-be-linked
```

LINK command qualifiers that may be useful are:

```
/DEBUG      - use when compiled with /DEBUG
/LIBRARY     - means file is a library
/INCLUDE=(module-list) - implies that qualified file is a library
/SHAREABLE  - for creating shareable images
/NOSYSSHR   - recommended but should not be necessary
```

CHAPTER 5

SYSTEM DEVELOPMENT

5.1 THE EBUILD COMMAND

[UG:3-1]

The EBUILD command allows the user to build a VAXELN system from prepared images. These images may be user-written or supplied by DEC with your VAXELN toolkit.

```
$ EBUILD qualifier-list data-file-specification
```

The default output file type is .SYS and this contains the VAXELN system. The default qualifiers in effect render the EBUILD command:

```
$ EBUILD /EDIT /KERNEL=ELN$:KERNEL.EXE /LOG /NOMAP -  
_$ /SYSTEM=datafile-name.SYS
```

Chapter 13 of the VAXELN User's Guide is definitive on qualifiers. If a particular set of qualifiers is required routinely then the EBUILD command might be redefined in a LOGIN.COM file. For example:

```
$ !      Redefinition of EBUILD  
$ EB*UILD == "EBUILD /MAP /FULL"  
$ !
```

SYSTEM DEVELOPMENT

The full list of EBUILD qualifiers is:

<u>Qualifier</u>	<u>Negative form</u>	<u>Default</u>
BRIEF (works with /MAP)	NOBRIEF	BRIEF
EDIT	NOEDIT	EDIT
FULL	NOFULL	NOFULL
KERNEL	-	KERNEL=ELN\$:KERNEL.EXE
LOG	NOLOG	LOG
MAP=file-specification	NOMAP	NOMAP
SYSTEM=file-specification	-	SYSTEM=datafile-name.SYS

The results of your dialogue with the System Builder are recorded in a .DAT file. This file contains printable text of additions or changes made in an /EDIT session. You may create a new system from an existing .DAT file by simply issuing the EBUILD command thus:

```
$ EBUILD /NOEDIT file-name.DAT
```

The System Builder comprises a number of menus as described in detail in [HS:3-4 to 3-31]. The menus and the entries and defaults are listed below

5.1.1 Program Descriptions Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
Debug	Yes, No	No
Run	Yes, No	Yes
Init required	Yes, No	No
Mode	User, Kernel	User
User stack (initial)	1 - 32,767 pages	1 page
Kernel stack	1 - 32,767 pages	1 page
Job priority	0 - 31	16
Process priority	0 - 15	8
Job port message limit	0-16,384	16,384
Powerfailure exception	Yes, No	No
Argument(s)	In " "	None

5.1.2 Device Descriptions Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
Name	Device controller name	-
Register address	Physical 18-bit address (see table [HS:3-20])	-
Vector address	Address of device's first interrupt vector (see table [HS:3-20])	-
Interrupt priority	4 - 7	5
Autoload driver	Yes, No	Yes

SYSTEM DEVELOPMENT

5.1.3 System Characteristics Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
System image	None	-
Debug	Local, Remote, Both, None	Remote
Console	Yes, No	Yes
Instruction emulation	String, Float, Both, None	String
Boot method	Disk, ROM, Down-line	Down-line
Disk/volume names	Device and volume info	-
Guaranteed image list	Shareable images, separated by ','	-
Page table slots	2 - 32,767 (2 per job 1 per subprocess)	64
Ports	2 - 32,767	256
Pool size	16 - 32,764 blocks	384
Virtual size	128 - 32,640 pages	1,024
Interrupt stack	2 - 8,192 pages	2
I/O region size	0 - 32,767 pages	128
Dynamic program space	0 - 32,767 pages	0
Time interval	1 - 120,000,000 microseconds (2 min)	10,000 microseconds (fixed on MicroVAX)
Connect time	1 - 3,599 seconds	45 seconds
Memory limit	0 - 65,535 pages	0 (use all available)

5.1.4 Network Node Characteristics Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
Network Service	Yes, No	Yes
Name server	Yes, No	Yes
File Access Listener	Yes, No	Yes
Network device	UNA, QNA, Other	QNA
Node name	1 - 6 chars (not needed for down-line load)	-
Node address	Not needed for down-line load	0
Authorization required	Yes, No	No
Authorization service	Local, Network, None	None
Authorization file	File-spec	AUTHORIZE.DAT
Default UIC	Valid UIC	[1,1]
Node triggerable	Yes, No	Yes
Network segment size	192 - 1,470 bytes	576 bytes

SYSTEM DEVELOPMENT

5.1.5 Terminal Descriptions Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
Terminal	Only if terminal required	-
Terminal type	Controller - DMF, DZ, DH	DZ
Speed	Range from 50 - 9600 baud	9600
Parity	Yes, No	No
Parity type	Odd, Even	Even
Display type	Scope, Hardcopy	Scope
Escape recognition	Yes, No	Yes
Echo	Yes, No	Yes
Pass all	Yes, No	No
Eight-bit	Yes, No	No
Modem	Yes, No	No
DDCMP	Yes, No	No

5.1.6 Console Characteristics Menu

<u>Entry</u>	<u>Responses</u>	<u>Default(s)</u>
Display type	Scope, Hardcopy	Hardcopy
Escape recognition	Yes, No	Yes
Echo	Yes, No	Yes
Pass all	Yes, No	No
Eight-bit	Yes, No	No

Example of brief and full maps for a simple program:

```
----- PROGRAM -----  
  
MODULE Simple [IDENT ('V1.000')];  
  
PROGRAM Simple (OUTPUT);  
  
    BEGIN  
        WRITELN ( 'A simple program to test VAXELN' );  
    END { of PROGRAM }.  
  
END { of MODULE };  
  
----- EBUILD .DAT FILE -----  
  
characteristic /noconsole  
program SIMPLE /debug  
  
----- EBUILD COMMAND -----  
  
$ EBUILD /NOEDIT /MAP /BRIEF SIMPLE
```

SYSTEM DEVELOPMENT

----- MAP FILE -----

VAXELN System Builder	27-MAY-1987 12:30:02.15
ELN V2.3-00	27-MAY-1987 12:30:02.15

System file

SIMPLE	DISK\$INSTRUCT:[SHONE.VAXELN]SIMPLE.SYS;1
--------	---

Kernel

KERNEL	SYS\$SYSDEVICE:[ELN]KERNEL.EXE;3
--------	----------------------------------

Programs

XQDRIVER	SYS\$SYSDEVICE:[ELN]XQDRIVER.EXE;3
EDEBUGREM	SYS\$SYSDEVICE:[ELN]EDEBUGREM.EXE;3
SIMPLE	DISK\$INSTRUCT:[SHONE.VAXELN]SIMPLE.EXE;1

Devices

XQA	
-----	--

Terminals

Shareable images

NETWORK	SYS\$SYSDEVICE:[ELN]NETWORK.EXE;3
PASCALMSC	SYS\$SYSDEVICE:[ELN]PASCALMSC.EXE;3
DAP	SYS\$SYSDEVICE:[ELN]DAP.EXE;3
PRGLOADER	SYS\$SYSDEVICE:[ELN]PRGLOADER.EXE;3
DPASCALIO	SYS\$SYSDEVICE:[ELN]DPASCALIO.EXE;3
ELNACCESS	SYS\$SYSDEVICE:[ELN]ELNACCESS.EXE;2
VAXEMUL	SYS\$SYSDEVICE:[ELN]VAXEMUL.EXE;3

Network node characteristics

Network service	Yes
Name server	No
File access listener	No
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

System characteristics

Debugger Remote
Console driver No
Instruction emulation String
Boot method Downline-load
Volume/device names
Guaranteed image list
Page table slots 64
Ports 256
Pool size 384 blocks
Virtual size 1024 pages
Interrupt stack 2 pages
I/O region size 128 pages
Dynamic program space 0 pages
Time interval 10000 microseconds
Connect time 45 seconds
Memory limit 0 pages

System image size is 280 pages (140K bytes)

/NOEDIT/MAP/BRIEF SIMPLE

SYSTEM DEVELOPMENT

----- EBUILD COMMAND -----

\$ EBUILD /NOEDIT /MAP /FULL SIMPLE

----- MAP FILE -----

VAXELN System Builder
ELN V2.3-00

27-MAY-1987 12:30:19.34
27-MAY-1987 12:30:19.34

System file

SIMPLE DISK\$INSTRUCT:[SHONE.VAXELN]SIMPLE.SYS;2

Kernel

KERNEL SYSS\$SYSDEVICE:[ELN]KERNEL.EXE;3

(VAXELN kernel)

Vectors and Data: Start: 80000000 Pages: 4
Parameters: Start: 80000800 Bytes: 101
R/O Data and Code: Start: 80000868 Pages: 46
Transfer address: 00000000

Programs

XQDRIVER SYSS\$SYSDEVICE:[ELN]XQDRIVER.EXE;3

(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	50	NETWORK
Fixup vector	00008A00	1	

Transfer address: 00000644

EDEBUGREM

SYSS\$SYSDEVICE:[ELN]EDEBUGREM.EXE;3

(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

SIMPLE

DISK\$INSTRUCT:[SHONE.VAXELN]SIMPLE.EXE;1

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	
Read-only	00000400	1	
Fixup vector	00000600	1	

Transfer address: 0000041F

Devices

XQA

CSR address = %0774440

Vector = %0120

Priority = 4

BI number = 0

Adapter number = 0

Terminals

Shareable images

NETWORK

SYSS\$SYSDEVICE:[ELN]NETWORK.EXE;3

Major Id: 1, Minor Id: 0

Map into program region = Yes

Image section(s):

Type	Base VA	Page(s)
Read-only	8000C000	48

SYSTEM DEVELOPMENT

	Noshr Write	80012000	2
	Fixup vector	80012400	1
PASCALMSC	SYSS\$SYSDEVICE:[ELN]PASCALMSC.EXE;3		
	Major Id: 1, Minor Id: 3		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	80012800	9
DAP	SYSS\$SYSDEVICE:[ELN]DAP.EXE;3		
	Major Id: 1, Minor Id: 1		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	80013A00	51
	Fixup vector	8001A000	1
PRGLOADER	SYSS\$SYSDEVICE:[ELN]PRGLOADER.EXE;3		
	Major Id: 1, Minor Id: 0		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	8001A200	8
	Fixup vector	8001B200	1
DPASCALIO	SYSS\$SYSDEVICE:[ELN]DPASCALIO.EXE;3		
	Major Id: 1, Minor Id: 1		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	8001B400	38
	Fixup vector	80020000	1
ELNACCESS	SYSS\$SYSDEVICE:[ELN]ELNACCESS.EXE;2		
	Major Id: 1, Minor Id: 0		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	80020200	4
	Fixup vector	80020A00	1
VAXEMUL	SYSS\$SYSDEVICE:[ELN]VAXEMUL.EXE;3		
	Major Id: 1, Minor Id: 0		
	Map into program region = No		
	Image section(s):		
	Type	Base VA	Page(s)
	Read-only	80020C00	18

Network node characteristics

```

-----
Network service      Yes
Name server          No
File access listener No
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

```

System characteristics

```

-----
Debugger            Remote
Console driver       No
Instruction emulation String
Boot method          Downline-load
Volume/device names
Guaranteed image list
Page table slots     64
Ports                256
Pool size            384 blocks
Virtual size         1024 pages
Interrupt stack      2 pages
I/O region size      128 pages
Dynamic program space 0 pages
Time interval        10000 microseconds
Connect time         45 seconds
Memory limit         0 pages

```

System image size is 280 pages (140K bytes)

/NOEDIT/MAP/FULL SIMPLE

CHAPTER 6

BOOTING AND DOWNLINE LOADING

Once you have created a VAXELN system with the System Builder, the next stage in the development cycle is loading the system onto a target VAX. Two methods are available:

- from magnetic medium - disk or tape
- downline using an Ethernet

Part of the installation of VAXELN produces a system for testing purposes. It is called ICP.SYS and resides in the directory ELN\$. You might like to boot this system to ensure the installation was correct and also to practice the system loading sequences, before going live with your own.

6.1 BOOTING FROM DISK

[HS:4-1]

Booting your VAXELN system from disk assumes that the target machine is easily reached. The bootable image is transported on the chosen medium to the target. If the target is some distance removed from the host VAX a downline loading method might be more appropriate. See the section on downline loading for details of this method.

6.1.1 Making Bootable Media

The command procedure ELN\$:COPYSYS.COM allows you to create a bootable copy of your system (or the supplied ICP.SYS) on a:

- Files-11 disk or
- TU58 cartridge

BOOTING AND DOWNLINE LOADING

WARNING

Before attempting to make a bootable disk please ensure that you build your system with the Boot Method response as DISK in the System Characteristics menu. Failure to observe this setting will cause premature exit from ELN\$:COPYSYS.COM.

If you wish to make a bootable floppy disk, Application 9 in the Application Design Guide (ADG) page 9-1 lists the Pascal source for performing that function. The source for this and the other applications is supplied with your system. The details of using COPYSYS.COM and the console boot commands are at [HS:4-1].

6.2 DOWNLINE LOADING

Before downline loading may proceed both host and target VAXes must be configured correctly. It is assumed that communications hardware is installed on the host and target machines and that DECnet-VAX software is present on the former. If this is not the case, consult your system manager. Some familiarity with the Network Control Program (NCP) is desirable.

6.2.1 Validating The Host Node Database

Before a downline load request can be serviced the host node network data base must be updated. To see if the host node knows of the remote target, issue the following commands:

```
$ RUN SYS$SYSTEM:NCP                ( or $ MC NCP)
```

at the NCP> prompt issue:

```
NCP> SHOW NODE node-name CHARACTERISTICS
```

e.g. NCP> SHOW NODE GRUMPY CHARACTERISTICS

Node Volatile Characteristics as of 27-MAY-1987 13:21:25

%NCP-I-NMLRSP, listener response - Unrecognized component, Node

You or your system manager must supply the following details of the target machine to the network data base:

- o node address
- o node name

- o Ethernet hardware address
- o host VAX load device name

6.2.2 Node Address

This comprises an area code in the range 1-63 and a node number in the range 1-1023.

6.2.3 Node Name

This comprises 1-6 alphanumeric characters including at least one alphabetic character

6.2.4 Ethernet Hardware Address

This is printed on the controller board. For a MicroVAX proceed as follows:

- press the HALT button TWICE on the front panel of the MicroVAX. This produces the >>> prompt on the console
- examine the first DEQNA device register by typing: >>> E/P/W 20001920
- continue by issuing this command five times: >>>E +
- the six strings returned provide you with the details of the Ethernet hardware address. The output on the console might look something like this:

```
>>> E/P/W 20001920
P 20001920  FFAA
>>> E +
P 20001922  FF02
>>> E +
P 20001924  FF12
>>> E +
P 20001926  FF04
>>> E +
P 20001928  FF1B
>>> E +
P 2000192A  FF88
```

The last two characters on each string returned constitute the hardware address that you are seeking. From the example above the hardware address would be: AA-02-12-04-1B-88.

BOOTING AND DOWNLINE LOADING

6.2.5 Host VAX Load Device

Your host VAX load device will be a DEUNA and the service circuit UNA-0. For a MicroVAX host the load device would be a DEQNA and the service circuit QNA-0.

6.3 CONFIGURING THE HOST NETWORK DATA BASE

The DEFINE and SET commands recognized by NCP require SYSPRV and OPER privilege respectively. A system User Identification Code (UIC) will also permit use of the NCP command DEFINE. Your system manager is unlikely to allow you these levels of privilege unless you are an operator or you perform a system manager function. To configure your VMS host for downline loading you must 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
NCP> DEFINE NODE EXAMPL ADDRESS 4.22 SERVICE CIRCUIT UNA-0
NCP> DEFINE NODE EXAMPL HARDWARE ADDRESS AA-01-02-03-00-F2
NCP> SET NODE EXAMPL ALL
```

Circuits and Lines - MicroVAX

Where the host is a MicroVAX use QNA-0 in place of UNA-0 in the commands listed above

There are two data bases available for the network. They are the permanent and volatile data bases. The command "NCP> SET NODE EXAMPL ALL" above copies the permanent data base to the volatile data base. The latter will disappear when the host VAX is turned off but is restored from the permanent data base when the host is bootstrapped. The NCP command DEFINE adds data to the permanent data base while the SET command makes changes only to the volatile data base

6.4 CONFIGURING BOOTSTRAP LOADERS

The VAX-11/730 and 11/750 processors use the TU58 console storage medium for storing downline load bootstrap loaders. On MicroVAX processors these loaders are contained in the boot Read-Only Memory (ROM). VAXELN provides a command procedure - ELN\$:NEWBOOT.COM - to copy the bootstrap image file onto a TU58 console tape and, on 11-730 processors, a bootstrap command procedure. NEWBOOT.COM prompts for information before performing its tasks. Because NEWBOOT.COM writes to the console storage device - and hence requires the presence of the storage device's driver - the privilege CMKRNL is required. DEC recommends that NEWBOOT.COM is executed from the system manager account.

MicroVAX II language prompt

Where the host is a MicroVAX II part of the power-up sequence may include a request for the language to be used. To disable this request, switch to no language inquiry by turning the knob on the KA-630 panel above the baud rate setting knob to its top position - an arrow.

6.5 DOWNLINE LOADING

Before downline loading can proceed please fulfil the following:

1. the target machine must be running
2. the host network data base must be informed of the system to be loaded

Use the following command to perform this operation:

```
NCP> SET NODE EXAMPL LOAD FILE -
      _DISK$DEVELOP:[VAXELN.SYSTEMS]SIMULATOR.SYS
```

Note that .SYS is the default file type for a load file specification. You may wish to perform the same operation using EDEBUB thus:

```
$ EDEBUB /LOAD=DISK$DEVELOP:[VAXELN.SYSTEMS]SIMULATOR EXAMPL
```

Using EDEBUB requires OPER privilege as it performs an NCP SET operation

Boot method

Ensure that you specify Downline in the System Builder menu Edit System Characteristics, for the entry Boot Method, before building your system. Downline is the default.

The downline load bootstrap loader can be started using the console boot command "B" from the target machine. The commands for 730's, 750's and MicroVAX are:

- o 11/730 - >>> B XE0 (DEUNA loader)
- o 11/750 - >>> B DDA0 (Console storage)
- o MicroVAX - >>> B XQA0 (DEQNA loader)

At this point the host system receives a load request and the network software there creates a process called Maintenance Operation Monitor (MOM) to read the system image file specified in its network data base and sends it to the remote target's bootstrap loader. When using the downline load method for booting a VAXELN system you don't need to specify either the node name or node address in the System Builder menu Network Node Characteristics. These data are provided as part of the loading procedure. This has the advantage that the same system

BOOTING AND DOWNLINE LOADING

image may be loaded on several processors regardless of their name or address.

6.6 RELOADING TARGETS THAT HAVE NETWORK SERVICE

Provided your VAXELN system has the Network Service - by default it will have (see System Builder section) - you should not have to go to the target VAX to reboot it. To enable remote "triggering" set the default bootstrap loader to downline load by setting the default bootstrap selection switches to the correct read-only loader. For the three machines mentioned above the procedure is:

- o 11/730 - setting is performed by the NEWBOOT procedure
- o 11/750 - set default boot device switch to "A"
- o MicroVAX - set CPU configuration DIP switch 1 to "on" - this is the default.

The NCP command TRIGGER NODE node-name issues a boot request to the target and the VAXELN datalink device driver halts execution of VAXELN and initiates execution of the default bootstrap.

6.7 MONITORING NETWORK EVENTS

If you encounter problems on your network when loading systems downline the network event-logging facilities on the host will be useful in diagnosing problems. Your system manager/operator will help and may already have issued these commands:

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

Messages are written to the host system's console terminal. A user with OPER privilege may issue the DCL command:

```
$ REPLY /ENABLE=NETWORK
```

from any terminal and that will enable display of network event messages at that terminal. Messages like these will appear when downline loading occurs:

```

%%%%%%%%%% OPCOM 29-MAY-1986 09:20:26.14 %%%%%%%%%%
Message from user DECNET on S8VMS
DECnet event 0.3, automatic line service
From node 1.255 (S8VMS), 29-MAY-1986 09:20:25.93
Circuit UNA-0, Load, Requested, Node = 1.19 (ELNVAX)
File = SYS$SYSROOT:[SYSMGR]SIMPLE.SYS, Operating system
Ethernet address = AA-00-03-01-34-59

%%%%%%%%%% OPCOM 29-MAY-1986 09:20:34.70 %%%%%%%%%%
Message from user DECNET on S8VMS
DECnet event 0.3, automatic line service
From node 1.255 (S8VMS), 29-MAY-1986 09:20:34.65
Circuit UNA-0, Load, Successful, Node = 1.19 (ELNVAX)
File = SYS$SYSROOT:[SYSMGR]SIMPLE.SYS, Operating system
Ethernet address = AA-00-03-01-34-59

```

6.8 PROBLEMS WITH DOWNLINE LOADING

With a multi-node VAX environment it is possible to have two or more host VAX processors capable of responding to a target system's boot request. Ensure that only one host VAX can respond to a target system's boot request. If device timeout messages are being logged by DECnet-VAX it is well worth checking that only one node's data base contains load information for a specific target. If you are in doubt have the following commands executed:

on old node:

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

on new node:

```
NCP> SET NODE EXAMPL SERVICE CIRCUIT UNA-0 -
    _HARDWARE ADDRESS AA-01-02-03-00-F2
```

BOOTING AND DOWNLINE LOADING

6.9 PROBLEM BOOTING FROM DEQNA

If a console message like this should appear:

```
>>> B XQA0
..2
CTRLERR XQA0
.
.
Failure.
>>>
```

don't suspect the DEQNA immediately. Check the Ethernet transceiver connection is sound and that the connector to the MicroVAX into the DEQNA socket is firm.

CHAPTER 7

DEBUGGING

7.1 METHODS OF DEBUGGING

There are two methods for debugging a VAXELN system:

1. remotely
2. locally

7.1.1 Debugging Remotely

If you choose remote debugging the System Builder includes remote debugger software in the system. This allows you to debug your system from your host VAX via the Ethernet connecting host and target using the DCL command EDEBUG. The benefits of remote debugging are:

- o debug access to one or more VAXELN target systems at the same time
- o availability of the debug symbol table provided by compilers, giving access to variable names, labels and source-line information
- o your terminal acts as the console device on the target system

7.1.2 Debugging Locally

If you choose local debugging the System Builder includes the entire debugger utility in your system. Facilities available with the local debugger are:

- o allows you to debug your system from your target VAX
- o those facilities available when debugging remotely are unavailable when debugging locally

DEBUGGING

- o a network connection between host and target is not essential.
- o ability to debug the VAXELN kernel and processes on the running system

7.2 CHOOSING A DEBUGGING MODE

There are four options available on the Edit System Characteristics menu entry for Debug they are:

- o NONE - a fully debugged system not requiring the presence of debugging facilities
- o LOCAL - the local debugger is built into the system image
- o REMOTE - the remote debugger is built into the system and EDEBUG is used to access the target remotely over an Ethernet
- o BOTH - provides local and remote capabilities

Omission of the console device from your system will make your terminal the console device for the system when you establish a connection with EDEBUG.

7.3 KERNEL DEBUGGING

Should you require to debug the VAXELN kernel please refer to [HS:5-23].

7.4 PROGRAM DEVELOPMENT AND DEBUGGING

When you wish to use the debugging facilities those routines you wish to debug should be compiled with the qualifier /DEBUG and linked with the LINK qualifier /DEBUG . (See also chapter 4 of this guide)

7.5 EDEBUG COMMAND AND ITS QUALIFIERS

For downline loading a VAXELN system from a remote VAX host the command line looks like this:

```
$ EDEBUG /LOAD=DISK$ELNDISK:[SPECIAL.PROJECTS]NEW.SYS FRED
```

This command causes the system NEW.SYS to be loaded through the Ethernet from the directory [SPECIAL.PROJECTS] on device DISK\$ELNDISK to the target VAX node name FRED and to pass control to the remote debugger. A message will appear at the terminal, at the bottom of the display:

```
Edebug V2.2-00
Loading "FRED".
Connecting to "FRED".
```

If during a session the following messages appear:

```
EDEBUG-F-COMM_ERROR, A communications error has occurred.
%SYSTEM-F-THIRDPARTY, network logical link disconnected by a third party
```

or

```
EDEBUG-F-NO_CONNECT, Connection failure
%SYSTEM-F-THIRDPARTY, network logical link disconnected by a third party
```

Issue the EDEBUG command again WITHOUT the /LOAD qualifier:

```
$ EDEBUG FRED
```

To load and start the system across the Ethernet without intervention of the remote debugger use the /NODEBUG qualifier thus:

```
$ EDEBUG /NODEBUG /LOAD=DISK$ELNDISK:[SPECIAL.PROJECTS]NEW.SYS FRED
```

Using the unqualified EDEBUG command enables the user to debug a system that is running already. For example:

```
$ EDEBUG FRED
```

7.6 LOGICAL NAMES AND EDEBUG

Like the VMS symbolic debugger EDEBUG uses the logical names DBG\$INPUT and DBG\$OUTPUT for I/O. Initially these logical names are assigned to SYS\$INPUT and SYS\$OUTPUT. These may be useful when debug commands are to be read from a command procedure or debug output is preferred to another device.

7.7 EDEBUG FACILITIES

The facilities available from EDEBUG are similar to those of the VMS debugger. However instead of debugging just one piece of software in one job on one node debugging a VAXELN application may involve several jobs, processes and nodes. EDEBUG identifies the job, process and node in its session command prompt. The format of the prompt is:

```
Edebug JOB,PROCESS,NODE>
```

e.g. Edebug 7,3> meaning process 3 in job 7 is the target

DEBUGGING

An EDEBUG session can be in one of two states:

- o RUNNING - the process that is the target of the current session does not require attention
- o AWAITING COMMANDS - the current session is suspended awaiting commands from the terminal

Should you wish to change the target process of the current session you may use the SET SESSION command.

7.8 CONTROL-C SESSION

The Control-C session provided by EDEBUG permits the user to issue commands to the system rather than a particular process. It allows commands to be entered while a current session is active or when the current session is undefined i.e. no debugging sessions. The issue of a Control-C produces a new prompt thus:

EDEBUG CONTROL-C>

Using the Control-C facility does restrict the sorts of command that may be issued. Commands like Examine, Deposit, Go will NOT execute but several of the SHOW commands are valid as well as Evaluate and Halt commands.

7.9 EDEBUG COMMANDS

[HS:5-24 ->] carries a complete summary of the commands for use with EDEBUG with examples of their use. The list below provides a quick reference only

- o CALL target(argument-list) - invoke a routine and return to command mode when the routine completes
- o CANCEL BREAK address-expression - cancels a breakpoint at the address supplied. Also CANCEL BREAK /ALL and CANCEL BREAK /KERNEL
- o CANCEL CONTROL - processes start independently of the debugger in the current session's job
- o CANCEL EXCEPTION BREAK - reestablishes the default exception handler search
- o CREATE JOB program-name(argument-list) - creates a job running the program named
- o CREATE PROCESS target(argument-list) - invoke a routine in the context of the current program and start it as a process

- o CTRL/C - aborts current operation and gets the attention of the EDEBUG command interpreter
- o CTRL/Z - a synonym for EXIT
- o DEBUG nodename - debug, connect to or load and debug a system on another node. Also DEBUG /LOAD=system nodename
- o DEFINE identifier::type:=expression - create or redefine a session variable with specified type and initial value supplied by expression
- o DELETE PROCESS process-identifier nodename - delete a process, optionally on another node
- o DEPOSIT /qualifier address-expression := expression - deposit the value in expression in a location described by variable reference of address expression. Various data type qualifiers are allowed: ASCII, BYTE, WORD, LONGWORD, QUADWORD, REAL, FLOAT, DOUBLE, GRAND, D_FLOAT, G_FLOAT, HUGE, H_FLOAT.
- o EVALUATE expression - evaluate the expression. Also EVALUATE /BINARY expression, EVALUATE /HEX expression, EVALUATE /DECIMAL expression, EVALUATE /OCTAL expression, EVALUATE /ADDRESS adressexpression
- o EXAMINE qualifier-list address-expression - examine value in a location in the target system's memory. Location may be an address expression or a variable reference. Qualifiers for data types as for DEPOSIT. Also EXAMINE /INSTRUCTION address:address, EXAMINE /PSL, EXAMINE /SOURCE.
- o EXIT - Leave the debugging session
- o GO address-expression - continue with execution of the session
- o HALT process-specifier nodename - stops the current or specified process, optionally on another node
- o HELP - help for EDEBUG (stored in ELN\$:EDEBUG.HLB)
- o IF boolean expression THEN one-line-command - conditionally execute a command according to Boolean expression
- o LEAVE - leave execution of a substituted command
- o LOAD - install a new program image into the target system
- o PREDECESSOR expression - move scope of current session's reference a number of call frames backwards in the calling order
- o SEARCH /qualifier range target - search current program source for specified string or identifier. Qualifiers are /NEXT and /ALL.

DEBUGGING

- o SET BREAK /qualifier address-expression DO (one-line-command) - sets breakpoint at specified address. Qualifiers are /JOB, /ALL, /KERNEL
- o SET COMMAND identifier DO (one-line-command) - create a command
- o SET CONTROL - opposite of CANCEL CONTROL
- o SET EXCEPTION BREAK - stops associated session when any exception occurs and gives control to the debugger rather than initiating a search by the kernel for an exception handler. Cancelled with CANCEL EXCEPTION BREAK
- o SET LOG file-specification - causes logging of debug session. Omission of file-specification stops logging
- o SET MODE mode-name - alters several debugger command modes - modenames may be: DECIMAL, HEXADECIMAL, OCTAL, D_FLOAT, DOUBLE, G_FLOAT, GRAND, VERIFY, NOVERIFY, LINE, SOURCE, INTO, OVER, NOPROMPT.
- o SET PROGRAM image-file-specification - change session's program image
- o SET RETURN BREAK - stop session when current routine returns
- o SET SESSION /qualifier process-specifier nodename - change session to another, optionally on another node. Qualifiers are /GO and /KERNEL
- o SET STEP - change default stepping action. Options are INTO LINE, OVER LINE, INTO INSTRUCTION, OVER INSTRUCTION, INTO SOURCE, OVER SOURCE
- o SET TIME time-string nodename - set the system time on specified node using VMS time notation
- o SHOW BREAK - display breakpoint information
- o SHOW CALLS - display call history
- o SHOW COMMAND identifier - display one or all commands established with SET COMMAND. Optional qualifier /ALL
- o SHOW JOB job-id nodename - display information about all processes in a job. Job-id may be string, integer or identifier
- o SHOW MESSAGE expression - show text for value of an exit status
- o SHOW MODE - show current operating modes for the debugger
- o SHOW MODULE provide information about the program of the current session
- o SHOW PROCESS process-specifier nodename - display system state of a job or process, optionally on another node. Qualifier /ALL

- o SHOW PROGRAM name - show system information about installed program. Qualifier /ALL
- o SHOW SESSION process-specifier nodename - show debug state of one or all debugging sessions, optionally on another node. Qualifier /ALL
- o SHOW SYMBOL pathname identifier - displays information about a symbol. Qualifier /DEFINE
- o SHOW SYSTEM nodename - displays memory, CPU time and jobs on system, optionally on another node
- o SHOW TIME nodename - shows current system time, optionally on another node
- o SHOW TRANSLATION identifier OR string nodename - displays translation of a PORT object value, optionally on another node
- o STEP - execute next line or instruction. Also /INTO /LINE, /OVER /LINE, /INTO /INSTRUCTION, /OVER /INSTRUCTION, /INTO /SOURCE, /OVER /SOURCE
- o SUCCESSOR expression - move scope of current session's reference a number of call frames forwards in the calling order
- o TYPE module-name expression:expression - display source lines in range specified
- o UNLOAD program-name node - remove previously loaded program image, optionally from another node

7.10 SESSION LOGGING

It is useful to issue a SET LOG command for EVERY session so that progress may be reviewed. The commands issued may be repeated by using the log file as input to a future debugging session.

CHAPTER 8

PROCESSES, JOBS AND PROGRAM STRUCTURE

[LRM:2-1, RF:3-1]

8.1 JOB DEFINITION

A job executes a program image (produced by the VMS linker). Jobs are created automatically for specified program images when a system is started. The `CREATE_JOB` procedure enables run-time creation of jobs to run specific images that have been either included with a system or loaded with the dynamic program loader.

8.2 PROCESS DEFINITION

Processes are defined as the execution agents for VAXELN programs or for concurrently scheduled parts of programs. There is always a `MASTER` process that executes the program and zero or more subprocesses executing blocks of code called process blocks. Subprocesses are created dynamically by calling the VAXELN procedure `CREATE_PROCESS`.

8.3 PROGRAM DEFINITION

A program is the main routine of a job. There may be any number of jobs in a VAXELN system. Each job may run the same or different program images.

PROCESSES, JOBS AND PROGRAM STRUCTURE

8.4 ROUTINES IN VAXELN

A routine has two components:

- o a heading and
- o a body

8.4.1 Heading Of Routines

The type of a routine is indicated by its heading. A heading may also include parameters. Typical headings are:

- o PROGRAM
- o FUNCTION
- o PROCEDURE
- o PROCESS_BLOCK

8.4.2 Body Of Routines

The body of a routine is a block and has two components:

- o a series of declarations - constants, types, variables and other routines
- o in Pascal and C a compound statement delimited by BEGIN and END (Pascal) or { and } (C) containing the code to be executed when the routine is invoked

8.5 COMPILATION UNIT

For the purposes of design and development it is often convenient to write an application in small parts. These small parts are called compilation units in VAXELN and would exist in separate source files. By default, outer-level declarations in these compilation units are placed in the exported symbol table of an object module. An object module is created when a compilation unit is compiled using EPASCAL. A compilation unit consists of an outer-level routine declaration from one of the following:

- o PROGRAM block declaration
- o PROCEDURE declaration
- o FUNCTION declaration
- o PROCESS_BLOCK declaration
- o MODULE

The term module is applied to all compilation units in VAXELN the compilation of which results in a VMS object module (file of type .OBJ).

8.6 THE MODULE IN VAXELN

In VAXELN the reserved word MODULE is used to form a compilation unit that may contain:

- o module headers - EXPORT, IMPORT and INCLUDE
- o CONST declarations
- o TYPE declarations
- o VAR declarations
- o FUNCTION declarations
- o PROCEDURE declarations
- o PROGRAM block declaration
- o PROCESS_BLOCK declarations
- o INTERRUPT_SERVICE routine declarations
- o separate routine bodies - completing the definitions of routines whose declarations appeared elsewhere with the directive SEPARATE

PROCESSES, JOBS AND PROGRAM STRUCTURE

8.7 PROCESS STATES

[RF:3-3]

There are four process states:

- o RUNNING - process has control of CPU
- o READY - initial state of process. Run as soon as possible
- o WAITING - waiting for some condition(s) to be satisfied
- o SUSPENDED - has to be RESUMEd explicitly to return to READY

8.8 PROCESS PRIORITIES

[RF:3-4]

There are 32 levels of job priority and 16 for process priority:

	<u>Highest</u>	<u>Lowest</u>	<u>Default</u>
Job priority	0	31	16
Process priority	0	15	8

Job and process scheduling is on a preemptive basis.

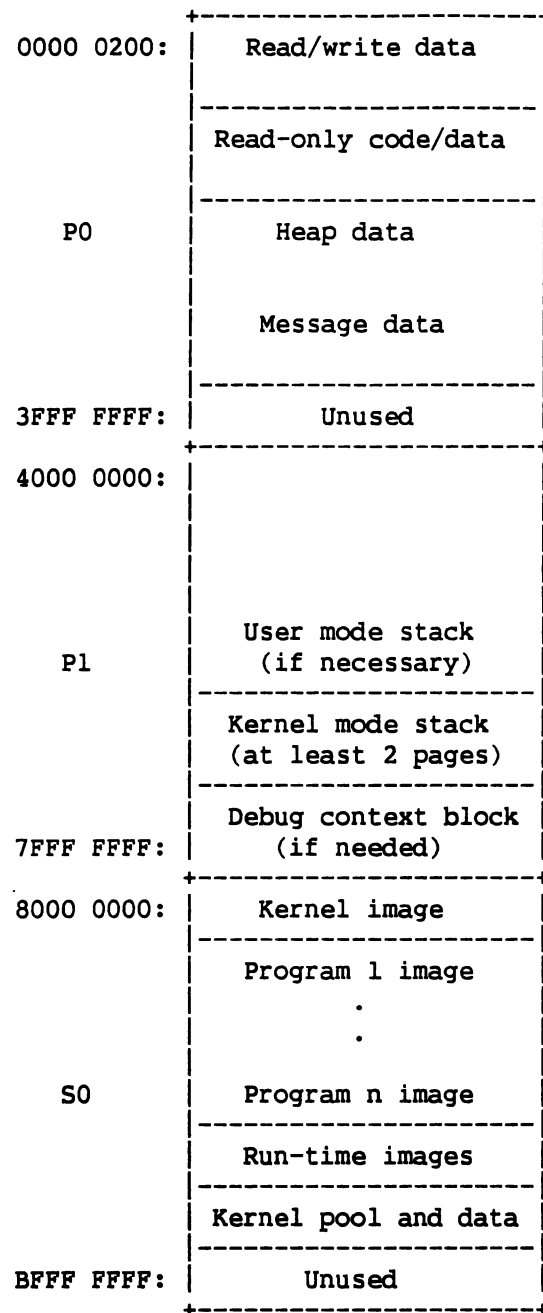
Two procedures for resetting priorities in jobs and processes are:

- o SET_JOB_PRIORITY
- o SET_PROCESS_PRIORITY

8.9 PROCESSES AND MEMORY MANAGEMENT

- o every job has a P0 and P1 page table
- o every process in a job shares the same master P0 region
- o each subprocess has a P1 page table

Layout of virtual memory



CHAPTER 9

TECHNIQUES OF SYNCHRONIZATION

[RF:4-1; LRM:11-1]

9.1 INTRODUCTION

The very nature of real-time renders synchronization a key issue in the development of such systems.

The sorts of problem that may arise are:

- o making things happen together
- o preventing things happening together
- o making things happen in order
- o making things happen in response to outside events

Lets look at these problems in a little more detail and see how VAXELN might solve them.

9.1.1 Making Things Happen Together

The filling of bottles with liquid on a production line requires that the bottles arrive beneath the fillers at precisely the moment when the liquid is emitted. A delay in either direction - early or late - results in spillage and loss.

9.1.2 Preventing Things Happening Together

Continuing our bottling plant example. Bottle capping must be prevented while liquid is being emitted into the bottles.

TECHNIQUES OF SYNCHRONIZATION

9.1.3 Making Things Happen In Order

In the previous example we might imagine a sequence of operations thus:

- o process A feeds empty bottles onto the line
- o process B controls emission of liquid into empty bottles
- o process C caps the filled bottles
- o process D labels the filled bottles
- o process E removes the completed bottle from the line

Clearly, to allow these processes to execute in any order will lead to failure rapidly. In VAXELN the use of WAITs for EVENTS or SEMAPHORES is a possible solution.

9.1.4 Making Things Happen In Response To Outside Events

In our bottling plant example, the exhausting of supplies of liquid would necessitate a halt of the bottling operation. Failure to achieve a satisfactory resolution of the problem would mean empty bottles with labels and caps! Here there is a need for the process monitoring liquid levels to be able to interrupt the process dispensing liquid or to interrupt the process that replenishes the liquid.

9.2 WAIT PROCEDURES

There are two wait routines in VAXELN:

- o WAIT_ANY
- o WAIT_ALL

Their call formats are:

```
WAIT_ALL    (  object_list,
               RESULT := wait_result,
               TIME    := tvalue,
               STATUS := stat
             )
```

```
WAIT_ANY    (  object_list,
               RESULT := wait_result,
               TIME    := tvalue,
               STATUS := stat
             )
```

The rules for using these procedures are:

- o both procedures may wait for 0 to 4 objects
- o objects for which the wait is being established may be:
 - AREA
 - DEVICE
 - EVENT
 - PORT
 - PROCESS
 - SEMAPHORE
- o WAIT_ANY is satisfied when any object for which it waits satisfies that wait
- o WAIT_ALL is satisfied when ALL objects satisfy the wait simultaneously
- o if the value in the wait_result is 0 the procedure timed out
- o if the value in the wait_result parameter for WAIT_ANY is 1-4 it signifies which object satisfied the wait. For WAIT_ALL the value is in the same range but is unpredictable
- o the time_value parameter enables a timeout option. Either an absolute time or delta time may be used. See time specifications later in this chapter. The wait is satisfied if the timeout expires.

9.3 SATISFYING A WAIT FOR KERNEL OBJECTS

- o AREA object - when the object is signaled or owner process of area is being deleted
- o DEVICE object - when the object is signaled
- o EVENT object - when the object is signaled
- o PORT object - when there is a message in the port
- o PROCESS object - when the process terminates (or is deleted)

TECHNIQUES OF SYNCHRONIZATION

- o SEMAPHORE object - when the object is signaled

9.4 PROCEDURES FOR SYNCHRONIZATION

In addition to WAIT_ALL and WAIT_ANY noted above the following lists the other synchronization-related procedures:

- o CREATE_EVENT
- o CLEAR_EVENT
- o CREATE_SEMAPHORE
- o DELETE
- o EXIT
- o SIGNAL

Their call formats are:

```
CREATE_EVENT    (    event,
                    initial_state,
                    STATUS := stat
                  )
```

```
CLEAR_EVENT    (    event,
                    STATUS := stat
                  )
```

```
CREATE_SEMAPHORE (    semaphore,
                    initial_count,
                    maximum_count,
                    STATUS := stat
                  )
```

```
DELETE         (    value,
                    STATUS := stat
                  )
```

```

EXIT      (  EXIT_STATUS := exit,
            STATUS      := stat
          )

SIGNAL    (  value,
            STATUS := stat
          )

```

9.5 THE MUTEX

[RF:2-13, LRM:11-56, Programs SYNCH_5.PAS, SYNCH_6.PAS]

Mutex stands for mutually exclusive semaphore. Using them improves efficiency when compared with using SEMAPHORES, WAITs and SIGNALs. A mutex is especially useful when two or more processes are trying to access a device e.g. a terminal. Application 8 supplied with your system demonstrates a mutex.

Mutexes are rather like gates. Locking a mutex (using procedure LOCK_MUTEX) closes the gate and increments the mutex count. The count starts at -1 when the mutex is created and goes to 0 when first locked. Further processes may lock the mutex, incrementing the count each time LOCK_MUTEX is called. If the counter is > 0 LOCK_MUTEX calls a WAIT procedure to wait for the semaphore.

When a process has finished with some resource it calls UNLOCK_MUTEX which decrements the counter. If the counter is >= 0 someone is waiting for the mutex and the binary semaphore is signalled and a process put into the wait state by LOCK_MUTEX can proceed to the resource. A simple but efficient metering facility.

The predeclared procedures to manipulate mutexes are:

- o CREATE_MUTEX
- o DELETE_MUTEX
- o LOCK_MUTEX
- o UNLOCK_MUTEX

TECHNIQUES OF SYNCHRONIZATION

9.6 SPECIFYING ABSOLUTE AND DELTA TIMES

The features of time on a VAXELN system are:

- o the internal representation of time on a VAXELN system is a 64-bit binary integer - LARGE_INTEGER data type is suitable for storing time values
- o the base date and time is: 17-NOV-1858 00:00:00.00
- o stored as the number of 100 nanosecond units since 17-NOV-1858 00:00:00.00
- o identical with VMS representation of time
- o available in two forms:
 - absolute
 - delta
- o routines available for manipulating time:
 - GET_TIME (equivalent to VMS SYS\$GETTIM) [LRM:9-60]
 - SET_TIME (equivalent to VMS SYS\$SETTIM) [LRM:9-61]
 - TIME_FIELDS (equivalent to VMS SYS\$NUMTIM) [LRM:9-62]
 - TIME_STRING (equivalent to VMS SYS\$ASCTIM) [LRM:9-64]
 - TIME_VALUE (equivalent to VMS SYS\$BINTIM) [LRM:9-66]

The details of the call formats for these services are at [LRM:9-59]

9.6.1 Format Of Absolute Time

Absolute time has the form:

dd-~~mmm~~-yyyy hh:mm:ss.cc

e.g. 13-JAN-1984 16:30:25.12

representing 25.12 seconds after 1630 hours on the 13th January 1984.

Note that:

- o there are 23 bytes in the string
- o month names are the first, or only, three letters of the month
- o month names may be uppercase or lowercase or a mixture of upper and lower cases
- o stored internally as a POSITIVE value

9.6.2 Format Of Delta Time

Delta time (or interval) has the form:

dddd hh:mm:ss.cc

e.g. 1061 23:15:18.09

representing 1,061 days 23 hours 15 minutes 18 seconds and 9 hundredths of a second

Note that:

- o there are 16 bytes in the string
- o stored internally as a NEGATIVE value

While there are shorthand methods of representing intervals it is recommended that strings are specified fully whenever possible. For example the string "0 :: 5" represents an interval of 5 seconds. The string "0000 00:00:05.00" also represents 5 seconds and is clearer from the documentation and maintenance viewpoint.

Appendix D of this guide contains specific examples of synchronization and calls to time routines.

CHAPTER 10

COMMUNICATION BETWEEN JOBS

[RF:5-1, LRM 12-1]

10.1 INTRODUCTION

Like human beings, VAXELN performs functions in jobs. Like human beings, jobs in VAXELN sometimes need to communicate information about their job (and the weather and last night's party).

In our working environments communication is sometimes local within the office or sometimes it extends beyond the bounds of our office to other offices in distant towns, cities and continents - that is, comparatively speaking, global.

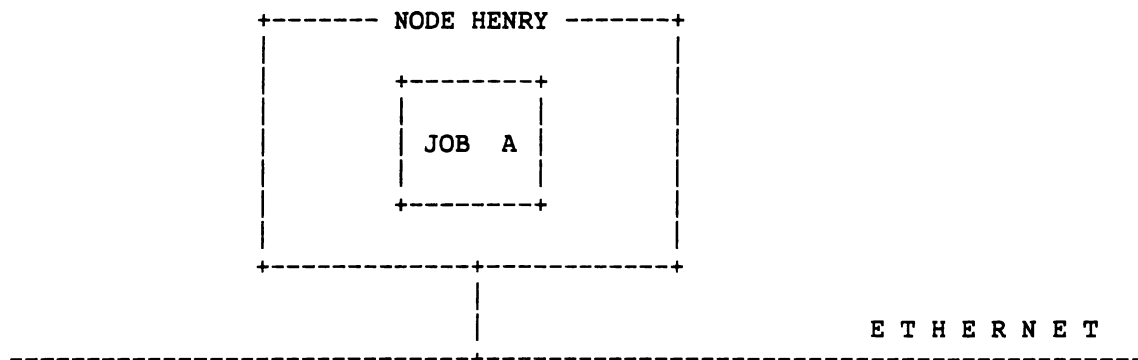
Communication between VAX jobs falls into two broad categories:

- o communication within one VAXELN processor - single node
- o communication between two or more VAX processors - multi-node

Note that we have not stated between VAXELN processors because we are not limited in VAXELN to communicating with just VAXELN nodes.

COMMUNICATION BETWEEN JOBS

10.2 COMMUNICATION WITHIN A SINGLE JOB ON A SINGLE VAXELN NODE



In the diagram above job A exists on a single VAXELN node. There are several methods for communicating within a job (intra-job communication) i.e. job A's master process talking with its subprocesses:

- o parameter passing on process creation
- o shared data item(s) - P0 shared between processes
- o shared AREA
- o sending MESSAGES

10.2.1 Parameter Passing On Process Creation

- o from 0 to 31 parameters
- o value or VAR
- o limited types:
 - ordinal types
 - pointer types
 - sets with 32 or less elements
 - REAL
 - AREA
 - DEVICE

- EVENT
- MESSAGE
- NAME
- PROCESS
- SEMAPHORE

10.2.2 Shared Data Item(s)

Data in scope in the master process - i.e. not data local to a procedure or function which goes onto the user stack - may be accessed from a subprocess block. Care should be exercised in allowing uncontrolled access. Synchronization is desirable to ensure correct order for readers and/or writers.

10.2.3 Shared AREA

A shared area is flexible because:

- o allows INTRA-job communication and
- o INTER-job communication

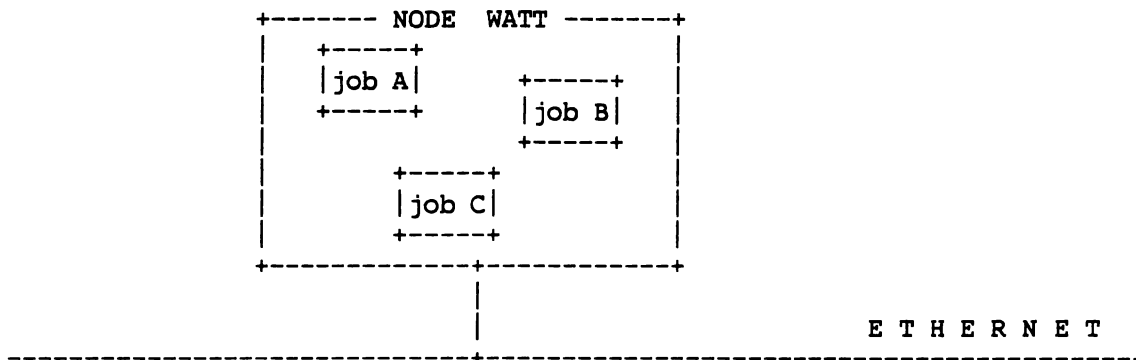
In other words a single job using an area for communication between it and its subprocess/subprocesses can be used by other jobs when the system develops/requires more than a single job. Please see discussion of areas in inter-job communication later in this section.

10.2.4 Sending MESSAGES

Usually not employed for communication between a master process and its subprocesses. The overhead of creating a message and additional ports means other alternatives listed above are preferable.

COMMUNICATION BETWEEN JOBS

10.3 COMMUNICATION BETWEEN MULTIPLE JOBS ON A SINGLE VAXELN NODE



In the diagram above jobs A, B and C exist on a single VAXELN node. There are several methods for communicating between jobs (inter-job communication) e.g. job A talking with job B:

- o argument passing on job creation
- o shared AREA
- o sending MESSAGES

No longer may P0 data be shared. At job level, memory management protects against access from external processes.

10.3.1 Argument Passing On Job Creation

When a job is created by the kernel or by a call to `CREATE_JOB` an optional argument list is available to the program running in that job's master process:

- o arguments are strings of up to 100 characters each
- o may be specified at:
 - system build time in Program Description menu
 - call to `CREATE_JOB` - these arguments override any specified with the system builder
- o two program argument functions are provided:
 - `PROGRAM_ARGUMENT` - result is program argument whose position is passed to the routine as an integer expression. The first position is 1.

- PROGRAM_ARGUMENT_COUNT - result is the number of program arguments. The routine has no arguments itself.
- o note that the program heading sometimes contains arguments: e.g. in Pascal: PROGRAM Driver_X (INPUT, OUTPUT); and additional file variables. Here program argument 1 is "INPUT" and program argument 2 is "OUTPUT"

10.3.2 Shared AREA

[LRM:12-26, Program COMM_5.PAS]

An area is:

- o a shareable region of memory
- o created and named using CREATE_AREA
- o associated with a binary semaphore
- o mapped in P0 virtual address space
- o physically contiguous 512-byte pages of memory

10.3.3 CREATE_AREA Kernel Service

[LRM:12-28]

```
CREATE_AREA      (  area,
                   data_pointer,
                   area_name,
                   VIRTUAL := base_va,
                   STATUS  := stat
                   );
```

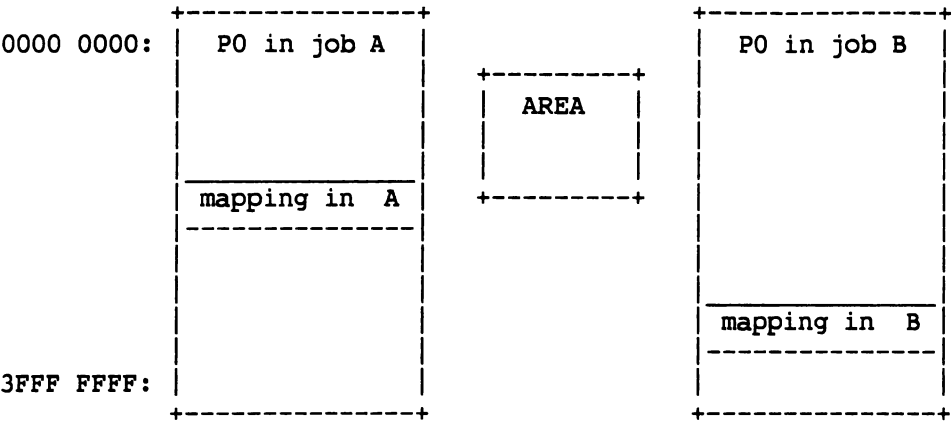
This kernel service either:

- o creates an new area or
- o maps an existing area

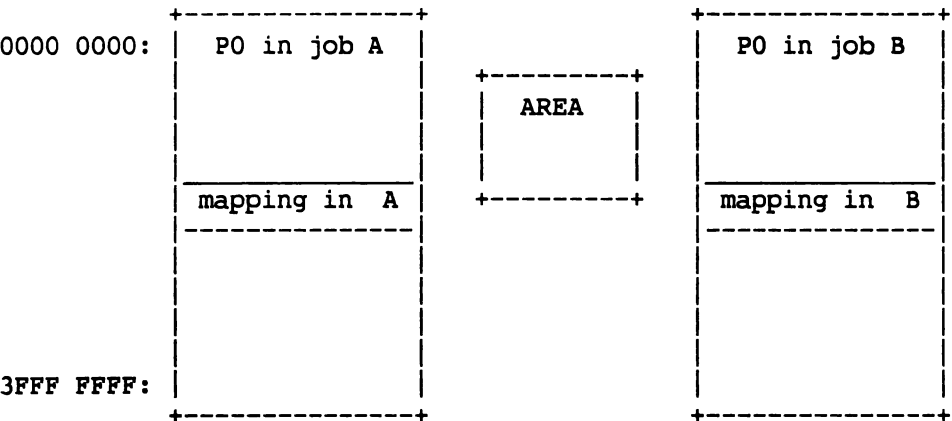
By specifying a base virtual address the area becomes position dependent. This means that accessing jobs can place pointer values in the area and those pointers will point to the same position in the P0 space of each job.

COMMUNICATION BETWEEN JOBS

Position Independent Area



Position Dependent Area



10.3.4 Sending MESSAGES

[RF:5-1] A message is:

- o a block of contiguous bytes of memory
- o created using CREATE_MESSAGE
- o mapped into job's P0 virtual address space (accessible to all processes in job if desired)
- o sent by unmapping, and received by remapping - no data copied
- o sent to and received from a PORT

10.3.5 CREATE_MESSAGE Kernel Service

[LRM:12-11, Programs COMM_*.PAS]

```
CREATE_MESSAGE (  message,
                  data_pointer,
                  STATUS := stat
                );
```

The data_pointer parameter cannot be a pointer to ANYTYPE. All other types are supported.

10.4 MESSAGE PORTS

A message port is like a maritime port - a place where carriers queue to unload cargoes. VAXELN's message ports are queues that contain messages

Important details about Ports:

- o created using CREATE_PORT
- o every job has a port by default. Its port value can be obtained by invoking the procedure JOB_PORT
- o port values may be:
 - sent in messages
 - passed as arguments
 - obtained from the RECEIVE procedure

COMMUNICATION BETWEEN JOBS

- used with WAIT_ALL and WAIT_ANY - thus waiting for a message provides an additional synchronization facility
- o ports have a limit on the number of messages that may be queued. This may be specified when the port is created
- o job port message limit specified at system build time through Edit Program Description menu (minimum 0, default and maximum is 16384)

10.4.1 CREATE_PORT Kernel Service

[LRM:12-15, Program COMM_7.PAS]

```
CREATE_PORT      (  port,
                   LIMIT  := int,
                   STATUS  := stat
                   );
```

The default value for LIMIT is 4.

10.4.2 Naming Message Ports

Ocean bound carriers know to which port they are destined by name. For example Amsterdam, New York, Tokyo, Singapore etc. In VAXELN too it makes life much simpler if ports are named. The procedure CREATE_NAME is provided for this purpose.

10.4.3 CREATE_NAME Kernel Service

[LRM:12-13, COMM_3.PAS]

```
CREATE_NAME      (  name,
                   name_string,
                   port_value,
                   TABLE  := table,
                   STATUS  := stat
                   );
```

The name_string parameter provides a 1 to 31 character string of the form of an identifier.

The optional TABLE parameter provides an enumerated value from the list of type NAME_TABLE:

- o NAME\$LOCAL
- o NAME\$UNIVERSAL
- o NAME\$BOTH

The default is NAME\$LOCAL. Without the Network Service in your system, names are placed in NAME\$LOCAL by default

The TRANSLATE_NAME procedure returns a port value for the port name passed to it.

10.5 SENDING AND RECEIVING MESSAGES

VAXELN provides two procedures for the transmission and receipt of messages:

- o SEND
- o RECEIVE

10.5.1 SEND Procedure

[LRM:12-21, Programs COMM_*.PAS]

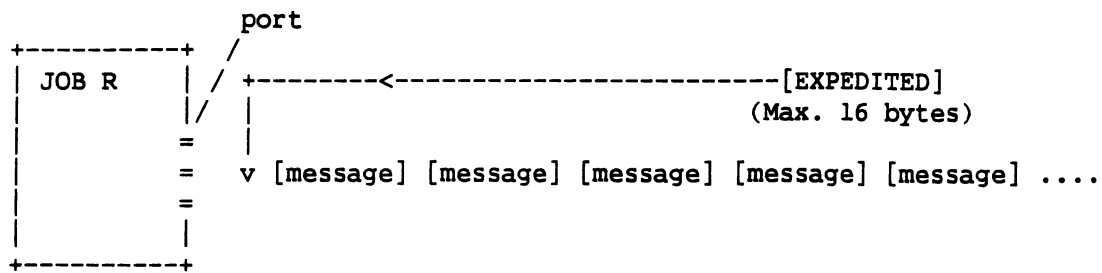
```
SEND      (  message,
             destination,
             SIZE      := size,
             REPLY     := reply_port,
             EXPEDITE  := expedite,
             STATUS    := stat
           );
```

The optional parameters, apart from STATUS are:

- o SIZE - indicates how many bytes of the message are to be sent. The default is all and the value if specified must be less than or equal to the size of the message data.
- o REPLY - provides a port value for replies. Default is sender's job port.
- o EXPEDITE - TRUE or FALSE indicating that the message is either expedited (TRUE) or a normal message (FALSE - the default)

Expedited messages are available - maximum length 16 bytes - to enable a sender to jump a queue of ordinary messages. Thus they are rather like a telegram - often short, usually cryptic - and get to the recipient quicker than normal mailing methods.

COMMUNICATION BETWEEN JOBS



10.5.2 RECEIVE Procedure

[LRM:12-19, Programs COMM_*.PAS]

```
RECEIVE (  message,
           data_ptr,
           port,
           SIZE      := size,
           DESTINATION := dest_port,
           REPLY      := reply_port,
           STATUS     := stat
        );
```

The optional parameters, apart from STATUS are:

- o SIZE - stores size in bytes of the message received
- o DESTINATION - normally value used by sender to send message
- o REPLY - provides a port value for replies. Not set properly if port is in a circuit

10.6 MESSAGE TRANSMISSION METHODS

There are two methods for transmitting messages:

- o using Datagrams
- o using Circuits

10.6.1 Using Datagrams

A process gets the value of a port and sends a message to that port. Probably

the only advantage of a datagram is it does not require circuits to be connected and accepted.

Disadvantages of datagrams:

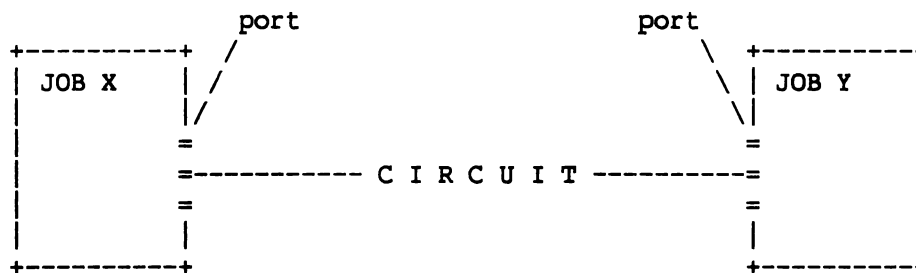
- o no guarantee of receipt at destination
- o no guarantee of order of messages
- o SEND returns failure if port is full
- o message may be lost if port full AND on another node

10.6.2 Using Circuits

Circuits are the recommended method for message transmission. There are several advantages of circuits

- o guarantees receipt at destination
- o guarantees order of messages
- o SEND can force waiting state if port full
- o can OPEN circuit like a file
- o messages are not lost if port is full

Two jobs can establish a circuit between them:



Let us assume that job X is expecting to receive a circuit connection from job Y. Job X executes a call to the ACCEPT_CIRCUIT procedure and enters the waiting state - waiting for job Y to issue a CONNECT_CIRCUIT request. Job Y is now able to gain control of the CPU (since job X is waiting) and issues a call to the CONNECT_CIRCUIT procedure. When the wait in job X is completed successfully the circuit is established between the two ports. For job X to send to job Y, job X prepares its message and SENDs to the port on its half of the circuit. The message is routed via the circuit to the port on job Y.

COMMUNICATION BETWEEN JOBS

NOTE: Issuing an ACCEPT_CIRCUIT call provides an additional synchronization tool.

10.6.3 ACCEPT_CIRCUIT Procedure

[LRM:12-6, Programs COMM_7.PAS, COMM_8.PAS and others]

```
ACCEPT_CIRCUIT      (  source_port,
                      CONNECT      := connect_port,
                      FULL_ERROR   := flag,
                      ACCEPT_DATA  := accdata,
                      CONNECT_DATA := conndata,
                      STATUS       := stat
                      );
```

The source_port parameter is the value of the port on which to wait for the connection request.

The optional parameters, apart from STATUS are:

- o CONNECT - an alternative connection port. Use this if you are establishing additional ports while others are still connected
- o FULL_ERROR - TRUE or FALSE. The default of FALSE implies that this process will wait on a SEND if the other port is full
- o ACCEPT_DATA - a VARYING_STRING of 16 bytes passed to the connecting process which receives it in the ACCEPT_DATA parameter in its call to CONNECT_CIRCUIT. This is like issuing a greeting on a handshake - "Hello I am X"
- o CONNECT_DATA - like ACCEPT_DATA except this string contains the connector's message to X - "Hello I am Y"

10.6.4 CONNECT_CIRCUIT Procedure

[LRM:12-8, Programs COMM_7A.PAS, COMM_8A.PAS and others]

```
CONNECT_CIRCUIT     (  port,
                      DESTINATION_PORT := dest_port,
                      DESTINATION_NAME := string,
                      FULL_ERROR       := flag,
                      CONNECT_DATA     := conndata,
                      ACCEPT_DATA      := accdata,
                      STATUS            := stat
                      );
```

The optional parameters, apart from STATUS are:

- o DESTINATION_PORT - destination port value for the connection request message. This may be omitted if a destination name is supplied.
- o DESTINATION_NAME - name of destination port for the connection. This argument is overridden by DESTINATION_PORT if it is supplied - use one or the other! This destination port need not be a VAXELN port e.g. it could be on a VMS node
- o FULL_ERROR - TRUE or FALSE. The default of FALSE implies that this process will wait on a SEND if the other port is full
- o CONNECT_DATA - a VARYING_STRING of 16 bytes that contains the connector's message to X - "Hello I am Y"
- o ACCEPT_DATA - a VARYING_STRING of 16 bytes that will take a message from the accepting process e.g. "Hello I am X"

10.6.5 Handling Full Ports - Flow Control

If ports are not connected (i.e. datagram method) a SEND to a full port returns an error status.

When the number of messages queued at a port connected in a circuit reaches the limit for that port - specified in the CREATE_PORT call or in the Program Description menu for Job Ports - two things can happen:

- o the sender process enters the waiting state or
- o the sender process receives an error status from the SEND procedure

The first outcome will arise if the FULL_ERROR argument was defaulted i.e. FALSE was implied, on a call to the sender's connect/accept circuit routine

The second outcome will arise if the FULL_ERROR argument was set TRUE

Clearly the default option - forcing a process into the waiting state - imposes a control on the flow of messages without having to program flow controls.

10.6.6 Communication Between Jobs On Multiple VAXELN Nodes

Notice that in the last diagram jobs X and Y were not shown to be existing on one particular machine. They could have been separated physically on two VAX processors with only an Ethernet between them.

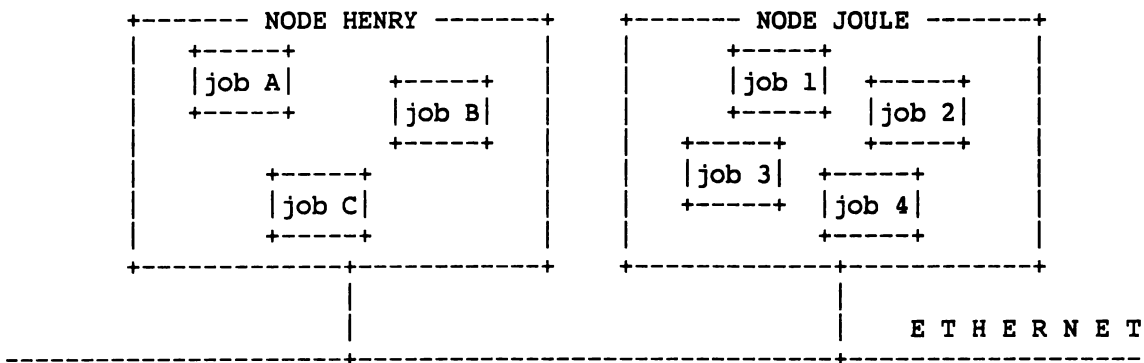
The very important messages here are:

COMMUNICATION BETWEEN JOBS

- o communication between jobs on one node and between jobs on several nodes is identical in programming terms
- o which node is running a particular job is irrelevant
- o communication between VAXELN nodes does NOT require node identification either as a number or as a name

In the diagram below we have bounded the jobs A, B and C within a node called HENRY - they reside in that processor's memory. Some physical distance away, but on the Ethernet with HENRY is node JOULE with jobs 1, 2, 3 and 4 resident in its memory. Communications (circuits) can exist between:

- o any job on node HENRY and any other job on node HENRY
- o any job on node HENRY and any job on node JOULE
- o any job on node JOULE and any other job on node JOULE
- o any job on node JOULE and any job on node HENRY
- o any job on node JOULE or node HENRY and a processor on the Ethernet



10.6.7 Disconnecting Circuits

Circuits are disconnected using the `DISCONNECT_CIRCUIT` procedure. This procedure takes a port value and, optionally, returns a status.

10.7 HINTS ON COMMUNICATION

1. use named ports
2. place port names in the universal name table
3. use circuits - ALWAYS
4. avoid expedited messages - careful design should mean a smooth and regulated flow of messages
5. remember the port message limit defaults to 4
6. agree/design message header protocols - the Pascal RECORD structure or the struct in C are ideal for message packets.

CHAPTER 11

NETWORK FACILITIES

[RF:7-1]

Network facilities on VAXELN are implemented by the Network Service. This service has the following capabilities:

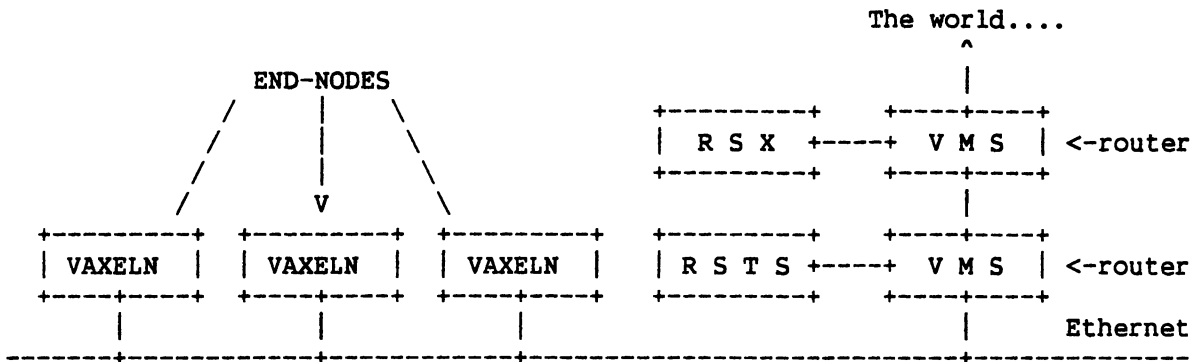
- o routes messages between network nodes
- o maintains a list of universal names
- o invokes the datalink driver to transmit messages
- o provides port name translations with the kernel
- o communicates with other DECnet nodes e.g. with EDEBUG from a VMS host
- o uses Phase IV DECnet protocols
 - Routing Protocol Version 2.0 end-node routing
 - Network Services Protocol (NSP) Version 4.0
 - Session Control Protocol Version 1.0

End-node routing means:

- o no network through-traffic can be handled from, say, a VMS node sharing the same Ethernet
- o a VAXELN system can communicate with ANY DECnet node on the Ethernet
- o a VAXELN system can communicate with ANY node in a network via a full routing node

NETWORK FACILITIES

End nodes and routing nodes



The Network Service is built into your system:

- o by default
- o when remote debugging is selected

The Network Service is not needed for intra-node communications and you should select "No" against Network Service in the Network Node Characteristics Menu of EBUILD, unless you are using remote debug.

11.1 LOCAL AND REMOTE LINKS WITH CIRCUITS

The same circuit services - `CONNECT_CIRCUIT` and `ACCEPT_CIRCUIT` - are used whether the connection is local or remote. The Network Service establishes an NSP logical link between the ports involved whether the remote port is VAXELN or non-VAXELN.

11.2 DECNET MESSAGE SIZE

Both DECnet and Ethernet impose a limit on message size. NSP calls this the segment size and this name appears in the Network Node Characteristics menu of EBUILD. The default size is 576 bytes with a minimum of 192 and maximum of 1470 bytes. From any of these values must be subtracted 32 bytes of header, so the smallest segment is 160 bytes and the largest 1438 bytes.

The value you choose should correspond with that for all your other VAXELN nodes as well as those VMS nodes with which you wish your VAXELN system to communicate. To find the Buffer Size value for your VMS node you must:

- o invoke NCP

- o enter the command SHOW EXECUTOR CHARACTERISTICS or
- o SHOW node-name CHARACTERISTICS

For example

NCP> SHOW EXECUTOR CHARACTERISTICS

Node Volatile Characteristics as of 27-MAY-1987 13:56:08

Executor node = 1.239 (S10VMS)

Identification	= DECnet-VAX V4.5, VMS V4.5
Management version	= V4.0.0
Incoming timer	= 45
Outgoing timer	= 60
NSP version	= V4.0.0
Maximum links	= 32
Delay factor	= 80
Delay weight	= 5
Inactivity timer	= 60
Retransmit factor	= 10
Routing version	= V2.0.0
Type	= routing IV
Routing timer	= 600
Broadcast routing timer	= 180
Maximum address	= 255
Maximum circuits	= 16
Maximum cost	= 1022
Maximum hops	= 30
Maximum visits	= 63
Maximum area	= 63
Max broadcast nonrouters	= 64
Max broadcast routers	= 32
Area maximum cost	= 1022
Area maximum hops	= 30
Maximum buffers	= 100
Buffer size	= 576 <<=====
Default access	= incoming and outgoing
Pipeline quota	= 3000
Default proxy access	= incoming and outgoing
Alias maximum links	= 32

NETWORK FACILITIES

NOTE

The segment size DOES NOT limit the size of your messages it simply indicates into what size packets your message will be segmented.

For example:

Let your message be 1800 bytes and segment size 576 (default)
Your message will require 4 segments thus:

where:

hd = header

----- E T H E R N E T -----																			
	hd		segment			hd		segment			hd		segment			hd		segment	
	32		544			32		544			32		544			32		168	

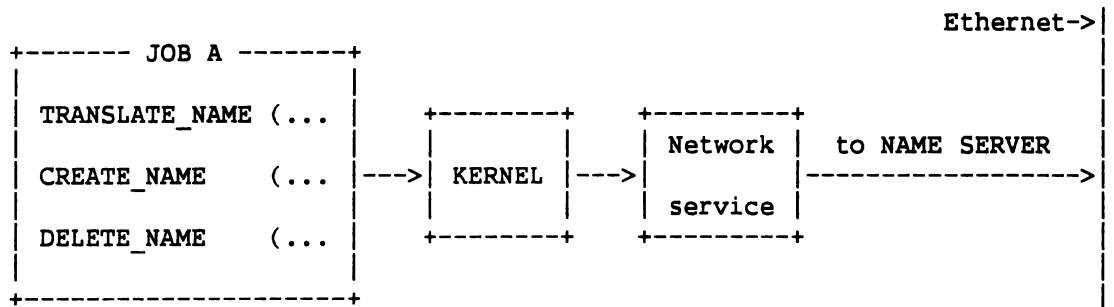
11.3 NAME SERVERS

[RF:7-6]

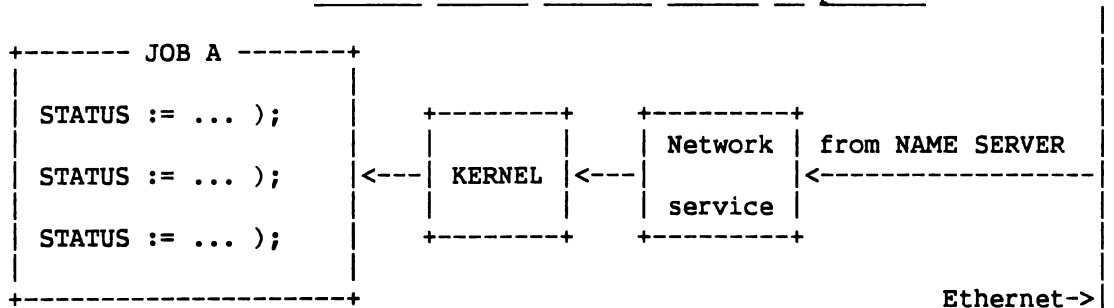
A name server is a VAXELN target responsible for maintaining the universal name list for the network. At any instant there is only one name server.

When processes create, delete and translate port names the local kernel communicates with its Network Service which in turn informs the network's name server. The name server acknowledges with a status message available in the optional STATUS parameter to CREATE_NAME, DELETE_NAME and TRANSLATE_NAME.

Name service requests sent to name server...



...name server returns status to process



NETWORK FACILITIES

11.3.1 Picking A Name Server

Each node keeps a list of universal names it has created even though it may not have the Name Server characteristic. This facilitates the selection of a current name server.

There are two instances when a name server has to be picked:

- o when the network starts
- o when the current name server fails

The node selected is:

- o the node with the highest Ethernet address

It performs its role:

- o by broadcasting its Ethernet address periodically
- o if this broadcast is not heard:
 - the node with the highest Ethernet address is elected name server
 - this node receives universal name lists from every other node

NOTE

DEC recommends that with less than 20 nodes in a VAXELN network, every node should take the default characteristic of Name Server. In the event of single node failures there is then an ample pool of volunteer name servers. A large number of name servers implies a large amount of network message traffic just to maintain the name service.

11.4 NODE IDENTIFICATION

- o VAXELN to VAXELN operations involving files or circuits DO NOT need node identification
- o VAXELN to VMS or VMS to VAXELN, for example, DOES require identification of nodes by one of two forms:
 - node name or

- node address

Please see chapter 6 for definitions of node name and node address.

The NCP command SHOW KNOWN NODES or the DCL command SHOW NETWORK reveal node names and addresses thus, for example:

\$ SHOW NETWORK

VAX/VMS Network status for local node 1.239 S10VMS on 27-MAY-1987 14:00:23.15

Node	Links	Cost	Hops	Next Hop to Node
1.239 S10VMS	0	0	0	(Local) -> 1.239 S10VMS
1.241 S4VMS	0	4	1	BNT-0 -> 1.241 S4VMS
1.249 M1VMS	0	9	2	BNT-0 -> 1.241 S4VMS
1.251 S6VMS	1	4	1	BNT-0 -> 1.251 S6VMS
1.254 S7VMS	0	5	2	BNT-0 -> 1.251 S6VMS
1.255 S8VMS	0	4	1	BNT-0 -> 1.255 S8VMS

Total of 6 nodes.

\$
\$

For example:

1.241 S4VMS

node name: S4VMS
node address: 1.241

which breaks down into:

area code: 1
node number: 241

11.4.1 Using Node Names

Use node names from a non-VAXELN system e.g. VMS. For example VAXELN node ELNODE might be interrogated thus from an adjacent VMS node:

\$ DIRECTORY /PROTECTION ELNODE::DISK\$ROBOTLOG:[ARMSLOG]*.MVT

11.4.2 Using Node Addresses

Use node addresses from VAXELN nodes to, say, a VMS node. For example VMS node

NETWORK FACILITIES

S4VMS might be interrogated thus from an adjacent VAXELN node:

```
OPEN ( VMSfile,  
      FILE_NAME := '1.241::DISK$PROJECTS:[ROBOTS]ARMS.DAT');
```

NOTE

VMS has no node name translation facility like that of VAXELN

11.5 MANAGING YOUR VAXELN NETWORK

There are two tools to help you look after your VAXELN network:

- o the Network Management Listener (NML) and
- o the Loopback Mirror

To use the VAXELN NML the node name and address of the VAXELN node must be present in the NCP database of the VMS node. The commands for achieving this were described in chapter 6.

To use the VAXELN NML for a VAXELN node called ELNODE, invoke NCP from VMS and issue these commands:

- o NCP> SET EXECUTOR NODE ELNODE
- o NCP> SHOW EXECUTOR

The output will look something like:

```
$ RUN SYS$SYSTEM:NCP  
NCP> SET EXECUTOR NODE ELNODE  
NCP> SHOW EXECUTOR
```

Node Volatile Summary as of 16-DEC-1986 10:45:55

Executor node = 1.376 (ELNODE)

```
State                = on  
Identification       = VAXELN V2.0
```

NCP>

Alternatively the following command achieves the same response:

```
$ RUN SYS$SYSTEM:NCP  
NCP> TELL ELNODE SHOW EXECUTOR
```

11.6 NCP COMMANDS SUPPORTED BY THE VAXELN NML

The following NCP commands are supported by VAXELN NML:

- o LOOP NODE node-id WITH data-type COUNT count LENGTH length

The parameters WITH, COUNT and LENGTH are optional and have the following possible values:

WITH : MIXED (default), ONES or ZEROS
COUNT : number of blocks to be sent, 1 to 65 535, default 1
LENGTH : length 1 to 65 535 bytes, default 40 bytes,
of blocks to be sent

- o SHOW EXECUTOR parameters

The parameters are:

SUMMARY, STATUS, CHARACTERISTICS or COUNTERS

- o SHOW KNOWN CIRCUITS parameters

The parameters are:

SUMMARY, COUNTERS

- o SHOW KNOWN LINES parameters

The parameters are:

SUMMARY, COUNTERS

- o SHOW NODE node-id parameters

The parameters are:

SUMMARY, COUNTERS

- o ZERO EXECUTOR

- o ZERO KNOWN CIRCUITS

- o ZERO KNOWN LINES

- o ZERO NODE node-id

The Loopback Mirror is useful for testing communication between host VMS system and a remote VAXELN node. For example, using ELNODE as our remote VAXELN node:

NETWORK FACILITIES

```
$ RUN SYS$SYSTEM: NCP
NCP> LOOP NODE ELNODE COUNT 100 LENGTH 50 WITH ZEROS
```

This sends 100 blocks of 50 bytes each containing binary zero to ELNODE

For testing communication between VAXELN node ELNODE and another VAXELN node called, say, ZENITH use the TELL command thus:

```
$ RUN SYS$SYSTEM:NCP
NCP> TELL ELNODE LOOP NODE ZENITH COUNT 20
```

This sends 20 blocks of 40 bytes each containing a combination of zeros and ones (defaulted on LENGTH and WITH) from ELNODE to ZENITH

11.7 CONNECTIONS TO AND FROM VMS

For full information please see the DECnet-VAX User's guide.

11.7.1 Connecting To VMS From VAXELN

Using the CONNECT_CIRCUIT procedure the parameter DESTINATION_NAME can take strings of this form:

'nodenumber::objectname' or 'nodenumber::objectnumber'

For example:

```
CONNECT_CIRCUIT (  myport,
                   DESTINATION_NAME := '7.347::STARTIT',
                   STATUS            := Return_status
                   );
```

This requests execution of STARTIT.COM in the default DECnet account on node 7.347 - area 7 node number 347.

or:

```
CONNECT_CIRCUIT (  my_other_port,
                   DESTINATION_NAME := '11.636::27',
                   STATUS            := Return_status
                   );
```

This requests execution of object 27 on node 11.636 - area 11 node number 636.

To see what objects are on your VMS system issue the NCP command SHOW KNOWN

OBJECTS.

From VMS the connection is achieved by using a file OPEN statement. This could be the DCL OPEN command or a high-level language version of OPEN, using the logical name SYS\$NET. The DCL command CLOSE or high-level language equivalent performs a DISCONNECT_CIRCUIT operation.

11.7.2 Connecting To VAXELN From VMS

From VMS use the language statement OPEN or SYS\$ASSIGN system service to make a connection. The name used should have the form:

```
'nodename::"TASK=portname"'
```

where:

nodename is the name of the VAXELN node
portname is the name of the VAXELN port

The VAXELN program needs to have a port and issue an ACCEPT_CIRCUIT call to complete the connection from VMS. If connecting to VAXELN by object number create a port name of the form:

```
'NET$OBJECT_objectnumber'
```


CHAPTER 12

USING FILES

[RF:9-1, LRM:15-1]

12.1 FILE ROUTINES

In addition to the standard file routines:

- GET
- RESET
- READ
- PUT
- REWRITE
- WRITE
- EOF

and for text files:

- EOLN
- PAGE
- READLN
- WRITELN

the following extensions are available in VAXELN Pascal:

- OPEN

USING FILES

- CLOSE
- FIND
- LOCATE
- FLUSH

and for text files.

- GET_CONTROL_KEY

The following file utilities are available [RF:9-7, LRM:15-55]:

- COPY_FILE
- CREATE_DIRECTORY
- DELETE_FILE
- DIRECTORY_CLOSE
- DIRECTORY_LIST
- DIRECTORY_OPEN
- PROTECT_FILE
- RENAME_FILE

Disk utility procedures available are [RF:9-11, LRM:15-73]:

- MOUNT_VOLUME
- INIT_VOLUME
- DISMOUNT_VOLUME

Tape utility procedures available are [RF:9-13, LRM:15-83]:

- MOUNT_TAPE_VOLUME
- INIT_TAPE_VOLUME
- DISMOUNT_TAPE_VOLUME

12.2 FILE OPENING AND CLOSING

There are two ways files may be opened (and closed):

- o explicitly - using OPEN (and CLOSE)
- o implicitly - using RESET, REWRITE. An implicit reset using READ or implicit REWRITE using PUT will achieve the same result.

NOTE

There is a cautionary note at [LRM:15-2] regarding closure of files. In particular the DELETE procedure when used with processes may result in buffered data to/from files being lost

12.3 INTERNAL AND EXTERNAL FILES

External files are associated with devices like disks or tapes.

Internal files are not associated with devices and may be used as structures in memory that may be accessed sequentially

12.4 FILE TYPES

Only one file type is supported currently:

- o SEQUENTIAL

12.5 FILE ACCESS

Two methods of file access are available:

- o sequential
- o direct (or random) - but records must be FIXED length

12.6 RECORD TYPES

Two types of record are supported:

- o fixed length
- o variable length

USING FILES

12.7 ENUMERATED TYPES USED IN FILE HANDLING

In addition to those predeclared enumerated types listed earlier in Chapter 2 there are the following:

[LRM:15-22]

Enumerated Type	Values
FILE\$PROTECTION_CATEGORIES	FILE\$SYSTEM FILE\$OWNER FILE\$GROUP FILE\$WORLD
FILE\$PROTECTION_TYPES	FILE\$DENY_READ_ACCESS FILE\$DENY_WRITE_ACCESS FILE\$DENY_EXECUTE_ACCESS FILE\$DENY_DELETE_ACCESS

12.8 FILE AND DEVICE SPECIFICATIONS

[RF:9-2, LRM:15-14]

For the file utility procedures listed above the limits on file specification are 1 to 255 characters

Storage devices for which drivers are supplied with VAXELN are listed in table 10-1 at [RF:10-3].

The format of a device name is:

- o device type
- o controller
- o unit number

For example device name: DUA2

DU	device type - e.g. RX50 floppy disk
A	controller A
2	unit number 2

This form of name is used in the Device Description menus in EBUILD

12.8.1 Specifying Volume Names

In the System Characteristics menu of EBUILD a volume name may or may not be specified for each device:

- o Specified - volume mounted automatically
- o Not specified - volume can be mounted dynamically using MOUNT_VOLUME - if it is a disk or MOUNT_TAPE_VOLUME - if it is a tape

The File Service will try to mount a volume in the drive if:

- o no volume name is supplied for the device
- o the volume name is not the same as the initialization name for the volume

In the latter case a console message will appear informing of this event.

A complete device-volume description might look like this in the System Characteristics menu of EBUILD:

```

      .      .      .      .      .
      .      .      .      .      .
      .      .      .      .      .

Instruction emulation  String  Float  Both  None
Boot method           Disk    ROM    Downline
Disk/volume names     "DUAL ERRLOG", "DUA2 DATA"

      .      .      .      .      .
      .      .      .      .      .
      .      .      .      .      .
  
```

The corresponding entry in the EBUILD .DAT file would look like this:

```
characteristic /volumes=("DUAL ERRLOG", "DUA2 DATA")
```

12.8.2 Default Volume

In the example above ERRLOG is the default volume for the File Service. If you don't specify a volume then the first volume mounted becomes the default volume

12.8.3 Volume Labels

Volume labels have the form:

USING FILES

- o DISK\$name - for disk volumes
- o TAPE\$name - for tape volumes

The name field MUST match the volume in the drive

12.8.4 Universal And Local Volume Names

Once a volume is mounted either

- o by the File Service or
- o by MOUNT_VOLUME or MOUNT_TAPE_VOLUME

the name - DISK\$name or TAPE\$name - becomes a UNIVERSAL name and identifies the volume from ANY local area network node.

However, if another volume is mounted with the same name as an existing universal volume name a LOCAL - local to that node - volume name is created. This would enable duplicate copies of data to held on volumes in the local area network.

Examples of the use of volume names

Let us assume our earlier definition:

Disk/volume names "DUA1 ERRLOG", "DUA2 DATA"

The volumes names are: DISK\$ERRLOG and DISK\$DATA

The default volume for the network is: DISK\$ERRLOG

```
OPEN ( Newfile,
        FILE_NAME := [IO_ERRORS]ERRORS.1985,
        HISTORY    := HISTORY$OLD );
```

opens an existing file on the default network volume. The full file specification would be DISK\$ERRLOG:[IO_ERRORS]ERRORS.1985

```
OPEN ( Newfile,
        FILE_NAME := DUA2:[IO_ERRORS]ERRORS.1985,
        HISTORY    := HISTORY$OLD );
```

opens an existing file on the LOCAL device DUA2.

```
OPEN ( Newfile,
        FILE_NAME := DISK$DATA:[IO_ERRORS]ERRORS.1985,
        HISTORY    := HISTORY$OLD );
```

opens an existing file on the network volume DISK\$DATA, and the local volume DISK\$DATA too, if one exists.

The rules are:

- o device name specified e.g. DUA2, the device is local to that node
- o device name specified e.g. DISK\$LOGGER, the device is a UNIVERSAL network volume and possibly if an individual node creates its own DISK\$LOGGER a local volume name too

This means that devices can be attached to just one node in a local area network and VAXELN systems have access, transparently, to devices (and files) on that node. Once again universal names greatly facilitate programming in VAXELN.

12.9 REMOTE FILE ACCESS

There is no requirement to hard-code node names for accessing files remotely. As discussed above the universal name concept applies to devices and volumes as it does to ports.

The File Access Listener (FAL) is responsible for connection requests to other

USING FILES

network nodes. These connections might be:

- o to open a file
- o to execute a command procedure on a VMS node

A VMS node can access files on a VAXELN node - authorization permitting (see Chapter 14). The files maintained and manipulated by VAXELN's File Service have the same structures as their disk and tape counterparts on VMS.

Many of the VMS file utilities may be used remotely. For example, using our VAXELN node ELNODE cited earlier with device specifications "DUAL ERRLOG", "DUA2 DATA", from a remote VMS node:

```
$ SEARCH /NOHEADER /LOG -  
_ $ ELNODE::[1985_DATA]CONSOLE_LOG.DAT "Message number"
```

This references DISK\$ERRLOG - the default for the node

```
$ DIFFERENCES ELNODE::DUA2:[SYSTEM]TODAY.DAT;45 -  
_ $ ELNODE::DUA2:[SYSTEM]TODAY.DAT;44
```

This references the local device on ELNODE DUA2

```
$ EDIT ELNODE::DISK$DATA:[REPORTS]13_JANUARY_1985.DAT
```

This invokes the default editor from the VMS node to edit the file on the local area network node ELNODE. The file may well not exist on ELNODE but on the node acting as file server

CHAPTER 13

DEVICES, DRIVERS AND INTERRUPTS

[RF:6-1, 10-1, LRM:14-1]

13.1 INTRODUCTION

This chapter defines some of the terms used when considering devices and the software associated with them.

It is beyond the scope of this course to discuss the design, writing and implementation of a device driver.

13.2 DEFINITIONS

13.2.1 Device - Definition

The VMS Glossary defines a device as:

"The general name for any peripheral hardware connected to the processor that is capable of receiving, storing, or transmitting data.

Card readers, line printers and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices. Devices are not necessarily hardware".

13.2.2 Device Drivers - Definition

The glossary to "Writing a Device Driver for VAX/VMS" (April 1986) states that a device driver is:

DEVICES, DRIVERS AND INTERRUPTS

"The set of instructions and tables that handles physical I/O operations to a device".

13.2.3 Interrupts - Definition

The glossary to "Writing a Device Driver for VAX/VMS" (April 1986) states that an interrupt is:

"An event other than an exception, or a branch, jump, case or call instruction that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs".

This definition is in terms of assembly language but the driver sources supplied with VAXELN are in Pascal. That makes it easier for high-level language programmers to follow the code.

13.3 DRIVERS SUPPLIED WITH VAXELN

A directory listing of the ELN\$ directory will reveal source files for some of the device drivers supplied with VAXELN:

```
$ DIR ELN$:*DRIV*.PAS
```

```
Directory SYS$SYSDEVICE:[ELN]
```

BDDRIVER.PAS;1	DDDRIVER.PAS;3	DHVDRIIVER.PAS;3	DQDRIVER.PAS;3
DUDRIVER.PAS;3	DZVDRIVER.PAS;3	LCDRIVER.PAS;3	LPVDRIVER.PAS;3
MUDRIVER.PAS;3	XBDRIVER.PAS;1	XEDRIVER.PAS;3	XQDRIVER.PAS;3
YCDRIVER.PAS;3			

Total of 13 files.

Chapter 10 of the VAXELN Run-Time Facilities Guide describes the features of these drivers. The code for LPVDRIVER is a good start for getting a feel for what is involved in writing a device driver. Simple examples are presented in LRM Chapter 14.

13.4 INTERRUPT SERVICE ROUTINES

Just a cursory glance through the code LPVDRIVER.PAS reveals several occurrences of the word INTERRUPT.

In order that a device can receive attention when it has data to be handled it has to have the help of the VAXELN kernel. This is achieved by connecting an interrupt service routine (ISR) to a device interrupt. An ISR is a piece of

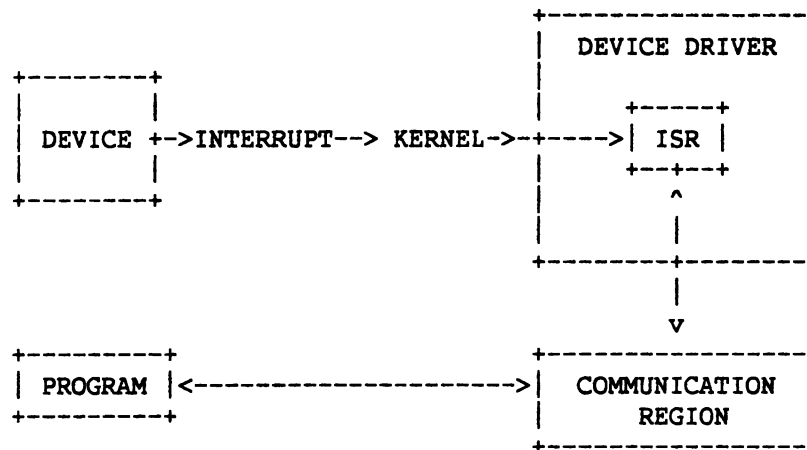
user-written code executed by the kernel. In some ways it is similar to the asynchronous systems traps of VMS - user-written code executed by the operating system.

So that a program may retrieve data from a device, an ISR can talk to a program via an area of memory called a communication region. This region is in S0 virtual address space (system)

ISRs can access data only as follows:

- o in locally declared variables
- o in the device's first control/status register (CSR), pointed to by the first parameter of the routine
- o in the communication region, pointed to by the second parameter of the routine

In pictorial form the components discussed above are arranged like this:



The connection of an ISR to an interrupt is made with the `CREATE_DEVICE` kernel service.

DEVICES, DRIVERS AND INTERRUPTS

13.4.1 Call Format For CREATE_DEVICE

The call format for CREATE_DEVICE is:

```
CREATE_DEVICE ( device_name,  
                device,  
                VECTOR_NUMBER      := relative_vector,  
                SERVICE_ROUTINE    := routine_name,  
                REGION             := region_pointer,  
                REGISTERS          := register_pointer,  
                ADAPTER_REGISTERS  := adapter_pointer,  
                VECTOR             := vector_pointer,  
                PRIORITY           := interrupt_priority,  
                POWERFAIL_ROUTINE  := power_routine,  
                STATUS             := stat  
            );
```

Note that provision of the name of an ISR (parameter SERVICE_ROUTINE) is optional.

An ISR would not be required if a device is to be polled. The technique of polling might be appropriate for a device like a thermometer where regular readings of the register or registers associated with the device, for display purposes, is all that is required.

13.5 DISABLING INTERRUPTS

It is important that a device should not interrupt while its own registers are being initialized. Disabling interruptions from a device or devices affects the way a VAX processor behaves. Processes calling DISABLE_INTERRUPT must be running in kernel mode.

13.6 INTERRUPT PRIORITY LEVELS

There are 32 levels of interrupt priority on a VAX processor with 0 being the lowest and 31 being the highest. The range 0 through 15 covers the software IPL's and 16 through 31 the hardware IPL's. A detailed list appears at [RF:6-3] in Table 6-1 which is reproduced below for convenience.

Interrupt Priority Levels**HARDWARE IPLs**
(decimal)**EVENTS**

31	Machine check; kernel stack not valid
30	Power failure
25-29	Processor, memory or bus error
24	Clock (except MicroVAX which is IPL 22)
16-23	Device IPLs, with 20-23 corresponding to UNIBUS or Q22 bus request levels 4-7 respectively

SOFTWARE IPLs
(decimal)**EVENTS**

9-15	Unused
8	DEVICE signal
7	Timer process
6	Queue asynchronous exception
5	Kernel debugger
4	Job scheduler
3	Process scheduler
2	Deliver asynchronous exception
1	Unused
0	User process level

To prevent interruption at a particular level raise the IPL to that level. For example raising IPL to 4, stops interruptions from that level and below.

The raising of IPL allows a process to synchronize itself with an ISR

13.7 UNEXPECTED INTERRUPTS - POWER FAILURE

Power failure is the most likely unexpected interrupt to occur to a VAXELN system. Because they are unpredictable these sorts of interrupt are classed as asynchronous. The penultimate parameter to the kernel procedure CREATE_DEVICE provides the option of specifying a powerfail recovery routine.

13.8 POWERFAIL RECOVERY ROUTINES

The important features of power recovery ISRs are:

- o they are called BEFORE the kernel restarts any other process or standard ISR
- o they must perform clean up operations like:

DEVICES, DRIVERS AND INTERRUPTS

- resetting the controller of the device - e.g. putting cylinder and track values back to a starting level on a disk controller
- prevent restart of part-finished I/O
- signal processes waiting for the device - no interrupts will occur after reinitialization

A look at the powerfail routine for LPVDRIVER shows the setting of a Boolean in the communications region indicating power failure and the signalling of the device:

```
.
.
.

{ Interrupt communication region }
comm_region = record
    powerfail    : boolean;
    busy         : boolean;
    lp_error     : boolean;
end; { record }

.
.
.

interrupt_service powerfail_recovery ( lpv11 : ^lpv11_registers;
                                       comm   : ^comm_region );
{++
{ powerfail_recovery - Powerfail recovery interrupt service routine
{
{   Inputs:
{
{       lpv11 - lpv-11 registers pointer
{       comm - Interrupt communication region
{
{   Outputs:
{
{       powerfail flag set
{--}

begin

with comm^ do
begin

    { If the device was busy, signal the waiting process }

    if busy then
        signal_device;
        powerfail := true;
    end;
```

end;

.
.
.

13.9 DEVICE INITIALIZATION ROUTINES

Your VAXELN development system includes a number of drivers for real-time devices. The initialization routines provided are:

- o AXV_INITIALIZE - for ADV or AXV analogue to digital converters
- o KVV_INITIALIZE - for KVV programmable realtime clocks
- o DLV_INITIALIZE - for DLV asynchronous serial line controllers
- o DRV_INITIALIZE - for DRV parallel line interfaces

Also included are READ and WRITE procedures for each of these devices. Details are at [LRM:14-41]

CHAPTER 14

SYSTEM SECURITY

[RF:8-1]

14.1 INTRODUCTION

The components of a VAXELN system that may require protection are:

- o the hardware
- o the software

14.1.1 Protecting The Hardware

Physical security of hardware may already be in place for existing equipment. Floppy disks and tape cartridges are small, compact items and easily removed in pockets or briefcases. Their removal may mean the departure of a complete and expensively-developed VAXELN system with the risk that the thieves may make copies for further gain. Even for VAXELN systems downline loaded the security of the master files on a host VAX must be considered.

14.1.2 Protecting The Software

Where downline loading is the preferred method there are potential security problems from other nodes on the network. Users on adjacent nodes may attempt to access data files stored on and manipulated by a VAXELN target machine. The consequences of a database or confidential material being read, corrupted or made inaccessible to the VAXELN target are obvious.

SYSTEM SECURITY

14.2 DEFAULT PROTECTION ON VAXELN

There is no protection on VAXELN. Because security may be more of a hindrance or a nuisance, when it is needed the appropriate EBUILD options must be specified explicitly.

14.3 VAXELN SYSTEM SOFTWARE PROTECTION

Apart from the VAX memory management facilities on VAXELN, the kernel does not impose control on programs. Some of the things that VAXELN does NOT do are:

- o control operating modes - kernel mode is freely available
- o limit priorities - programs can change priorities freely
- o limit process creation - no subprocess limits

VMS systems control access and resources by using privileges, quotas and limits. There are no such entities on a VAXELN system.

This freedom and accessibility may require VAXELN system designers and programmers to incorporate some authorization facilities similar to those on VMS. This will prevent the naive as well as the malicious gaining access to a VAXELN system and its resources

14.4 AUTHORIZATION FACILITIES

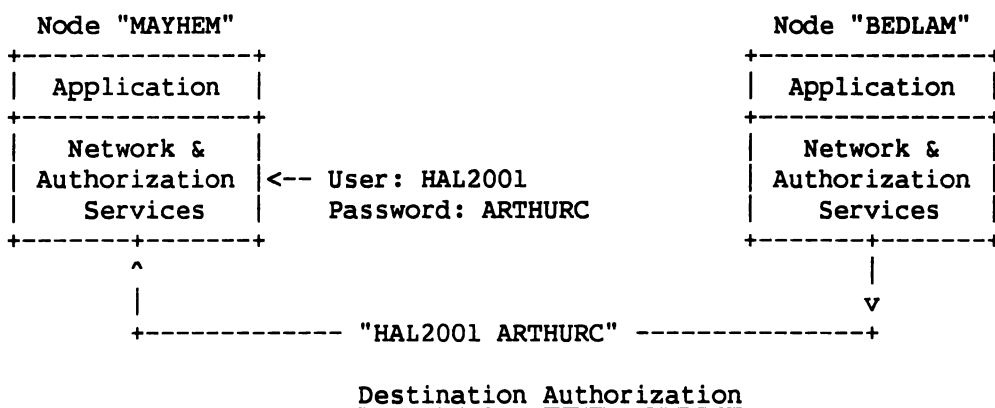
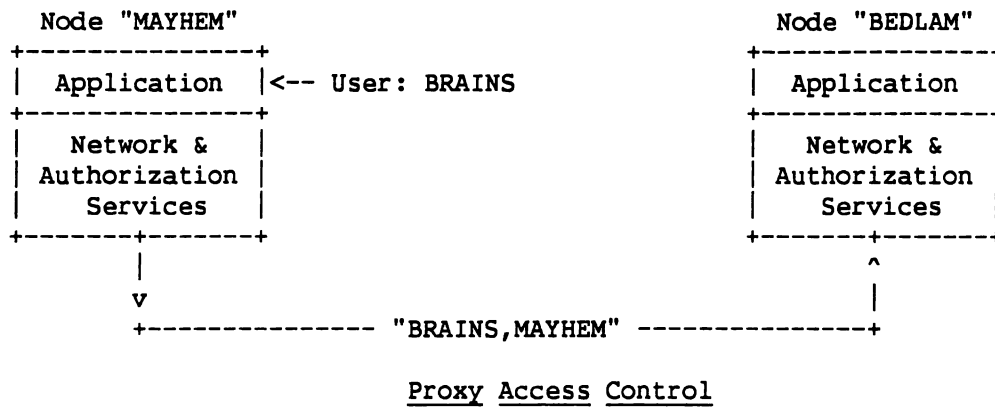
Building the authorization service into a system allows a database of users to be created and maintained. Those users are authorized to access the system having satisfied the authorization service of their identity.

14.4.1 Identifying Connection Requests

The authorization service identifies users in one of two ways:

- o user name and host node - proxy access control)
- o user name and password - destination authorization - using an access control string

The diagram below shows how the two access methods work



14.5 SPECIFYING REMOTE DESTINATIONS

14.5.1 Connecting To Non-VAXELN Nodes

For connection using object names the node number forms look like this:

- o 'nodenumber::objectname'
- o 'nodenumber"username password"::objectname'

SYSTEM SECURITY

- o 'nodenumber"username"::objectname
- o 'nodenumber"[ggg,mmm] password"::objectname

For connection using object numbers the node number forms look like this:

- o 'nodenumber::objectnumber'
- o 'nodenumber"username password"::objectnumber'
- o 'nodenumber"username"::objectnumber
- o 'nodenumber"[ggg,mmm] password"::objectnumber

14.5.2 Connecting To VAXELN Nodes

For connection using object names the node name forms look like this:

- o 'nodename::objectname'
- o 'nodename"username password"::objectname'
- o 'nodename"username"::objectname
- o 'nodename"[ggg,mmm] password"::objectname

For connection using object numbers the node name forms look like this:

- o 'nodename::objectnumber'
- o 'nodename"username password"::objectnumber'
- o 'nodename"username"::objectnumber
- o 'nodename"[ggg,mmm] password"::objectnumber

14.6 INCLUDING AUTHORIZATION IN A SYSTEM

The relevant network node characteristics page of EBUILD has this display:

System X - Editing Network Node Characteristics

```

.
.
Network device          UNA      QNA      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
.
.

```

14.6.1 Authorization Required

There are two possibilities:

- o Yes - means there will be no admission without authorization
- o No - means no authorization required. Free admission

The default is NO.

14.6.2 Authorization Service

There are three possibilities:

- o Local - authorization required for this system only
- o Network - authorization handler for whole network included
- o None - no service required

The default is NONE.

Only one node may be authorization server for a network and there must be at least one name server in the network too

SYSTEM SECURITY

14.6.3 Authorization File

The authorization file is AUTHORIZE.DAT by default but another file specification may be supplied if preferred. If the file does not exist one will be created.

14.7 USING AND MAINTAINING THE AUTHORIZATION SERVICE

When the authorization service is first established on a node only programs running on that node may manipulate the authorization file because the file will be empty. The following procedures are supplied for maintaining the authorization service:

- o ELN\$AUTH_ADD_USER - adds a record for a new user
- o ELN\$AUTH_MODIFY_USER - modifies the record for an existing user
- o ELN\$AUTH_REMOVE_USER - removes the record for an existing user
- o ELN\$AUTH_SHOW_USER - returns information about a user

These routines are described in detail at [LRM:11-40]. Currently the maximum lengths for authorization record fields are:

- o username - 20 bytes
- o password - 20 bytes - stored in a hashed/scrambled form
- o nodename - 32 bytes
- o userdata - 128 bytes - information to be stored with the other data

Only authorized users with a UIC group number of OCTAL 10 or less may manipulate the authorization database.

14.8 FILE SECURITY FACILITIES

Normal Files-11 protection facilities are available. Protection masks may be specified for whole volumes and defaults for files thereon when using INIT_VOLUME or for individual files using the PROTECT_FILE procedure

Example programs showing file handling will be found in Appendix G to this guide.

CHAPTER 15

HANDLING EXCEPTIONS

[RF:11-1, LRM:13-1, RTL:7-1]

15.1 INTRODUCTION

This chapter explains what is meant by the term exception and how VAXELN systems can handle different types of exception. The exception handling facilities provided by VAXELN are discussed as well as those for message handling.

15.2 DEFINITIONS

The VMS Glossary defines EXCEPTION as:

"An event detected by the hardware or software (other than an interrupt or jump, branch, case or call instruction) that changes the normal flow of instruction execution. An exception is always caused by the execution of an instruction or set of instructions (whereas an interrupt is caused by an activity in the system independent of the current instruction). There are three types of hardware exception: traps, faults and aborts. Examples are attempts to execute a privileged or reserved instruction; trace traps; compatibility mode faults; breakpoint instruction execution and arithmetic traps such as overflow, underflow and division by 0".

The VMS Glossary defines TRAP as:

"An exception condition that occurs at the end of the instruction that caused the exception. The program counter saved on the stack is the address of the next instruction that would normally have been executed. All software can enable and disable some of the trap conditions with a single instruction".

HANDLING EXCEPTIONS

The VMS Glossary defines FAULT as:

"A hardware exception condition that occurs in the middle of an instruction and leaves the registers and memory in a consistent state so that eliminating (the) fault and restarting the instruction will give correct results".

The VMS Glossary defines ABORT as:

"An exception that occurs in the middle of an instruction and sometimes leaves the registers and memory in an indeterminate state, which may prevent the instruction from being restarted".

Exceptions may be further categorized thus:

- o synchronous exceptions - happen at the same spot in a program given the same values, state etc.
- o asynchronous exceptions - unpredictable occurrence, the most obvious being power failure.

15.3 PROGRAM/PROGRAMMER RESPONSE TO EXCEPTIONS

A programmer has four options available when faced with a program that raises exceptions:

- 1 ignore the exceptions altogether
- 2 handle them with some additional code - called an exception handler
- 3 end the operation causing the exception and continue processing elsewhere
- 4 write an error message and terminate the execution of the program

An EXCEPTION HANDLER may be defined as:

"A procedure that the system executes when a process exception occurs. When an exception occurs, the operating system searches for an exception handler and, if found, initiates the handler immediately. The exception handler may perform some action to change the situation that caused the exception and continue execution for the process that incurred the exception. Exception handlers execute in the context of the process at the access mode of the code that incurred the exception".

(Based on the definition of Condition Handler in the VMS Glossary)

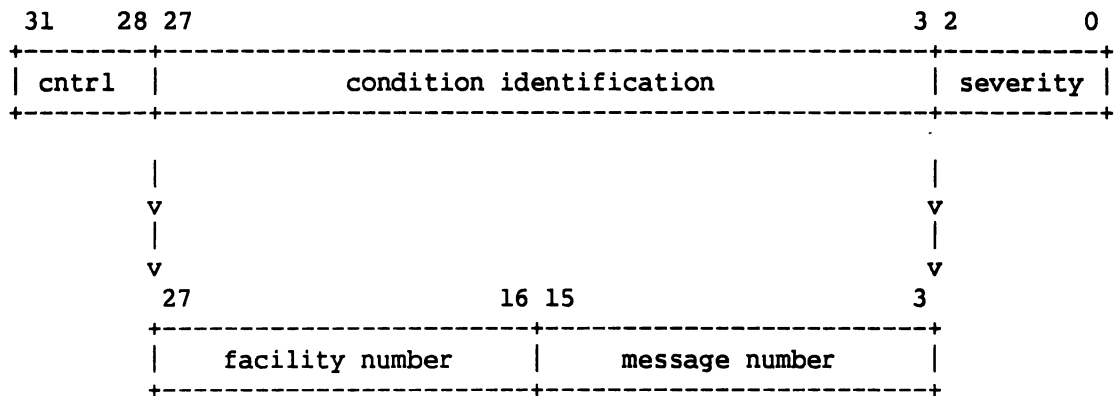
Note that the VAXELN kernel is responsible for invoking an exception handler not the user program. The RTL defines CONDITION as:

"An informational error state which exists when an exception occurs. The term condition is preferred since the term exception implies an error...".

The decisions about exceptions should be taken at the design stage of a piece of software. Consideration of the places in a program where exceptions may arise e.g. denominators of zero in divide operations, should not occur after they are discovered in the testing phase of a project - that is too late.

15.4 CONDITION VALUES

Every exception has a unique 32-bit condition value identifying the exception. The format of these 32-bit values is shown below:



HANDLING EXCEPTIONS

15.5 THE SEVERITY FIELD IN CONDITION VALUES

The severity values stored in the least significant three bits of the condition value range from 0 through 7 and have meanings as follows:

- 0 Warning
- 1 Success
- 2 Error
- 3 Information
- 4 Severe error
- (5 reserved to DIGITAL)
- (6 reserved to DIGITAL)
- (7 reserved to DIGITAL)

These severity codes, and others, are returned by VAXELN routines in the optional STATUS parameter to those routines. The codes conform to the same convention as VMS. That is:

- o ODD status values indicate SUCCESS
- o EVEN status values indicate some degree of FAILURE

15.6 STATUS CHECKING

It is essential that the optional STATUS parameter to VAXELN routine calls is used, especially while programs are under development and testing.

Failure to collect and test the status returned will mean time wasted searching for problems, bugs etc. Suitable checking code in Pascal might be something like this:

```
CREATE_PROCESS ( Process_identity,  
                Area_accessor,  
                STATUS := Returned_status );  
  
IF NOT ODD ( Returned_status ) THEN ....
```

Always check for specific status values using symbolic constants like KER\$_SUCCESS. DO NOT test for a value using integer constants. The following piece of code shows this bad practice:

```
IF Returned_status <> 1 THEN ...
```

Apart from the fact that maintenance programmers may not be familiar with a status of 1 the code is obscure and difficult to understand. More importantly there is more than one success code for some routines but all success codes have bit 0 set. DIGITAL recommends testing against either a specific symbolic code or for an ODD status value.

15.7 EVENTS WHEN EXCEPTIONS ARE RAISED

When an exception is signalled by a program the VAXELN kernel exception processing software:

- o stops normal execution of the program temporarily
- o calls an exception handler

If no user-defined exception handler is present then the kernel has two options:

- o if the debugger is in the system the special debugger handler is invoked
- o if the debugger is absent the process is deleted

How and where does the kernel find information about a user-defined exception handler?

15.8 THE STACK

The VMS Glossary defines STACK as:

"An area of memory set aside for temporary storage or for procedure and interrupt service linkages. A stack uses the LIFO concept. As items are added to ("pushed on") the stack, the SP decrements. As items are retrieved from ("popped off") the stack, the SP increments"

The stack pointer (SP - R14) is one of several important general registers used in handling procedures - and exception handlers - others are:

- o PC (R15) - the program counter
- o FP (R13) - current stack frame pointer
- o AP (R12) - argument pointer
- o R2 to R11 - saved registers

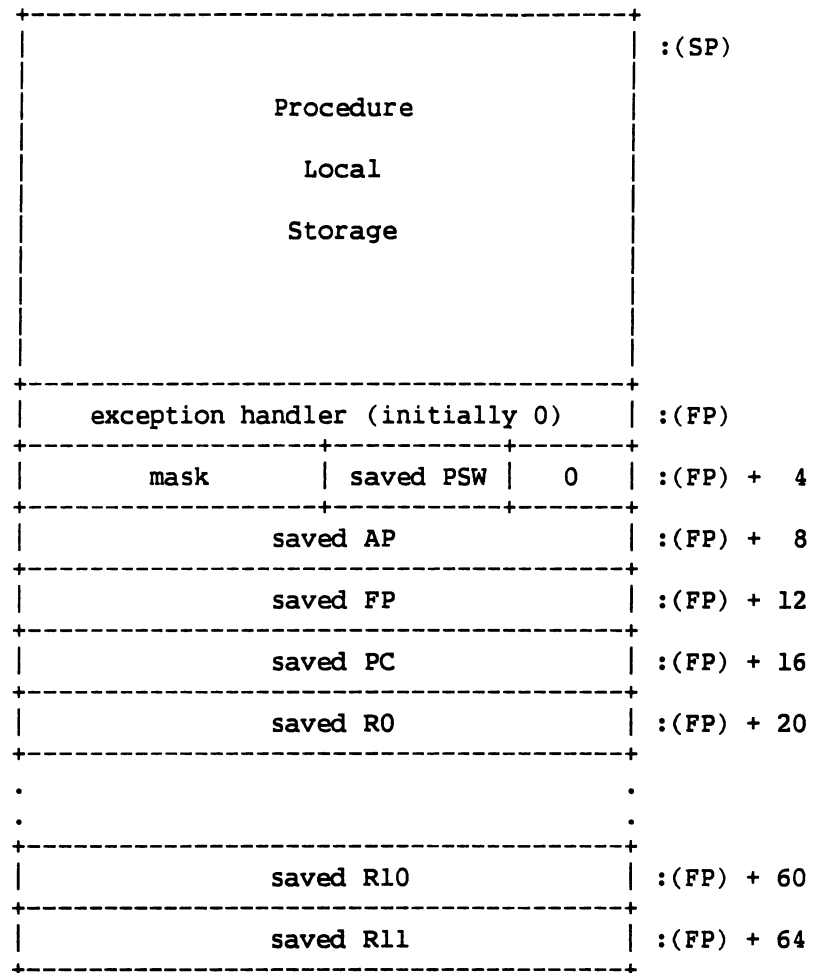
The stack is located in the high address end of P1 virtual memory. Each process has its own stack.

HANDLING EXCEPTIONS

A procedure that is executing has a call frame associated with it. The call frame looks like this:

exception handler (initially 0)				:(FP)
mask	saved PSW	0		:(FP) + 4
saved AP				:(FP) + 8
saved FP				:(FP) + 12
saved PC				:(FP) + 16
saved R0				:(FP) + 20
saved R1				:(FP) + 24
saved R2				:(FP) + 28
saved R3				:(FP) + 32
saved R4				:(FP) + 36
saved R5				:(FP) + 40
saved R6				:(FP) + 44
saved R7				:(FP) + 48
saved R8				:(FP) + 52
saved R9				:(FP) + 56
saved R10				:(FP) + 60
saved R11				:(FP) + 64

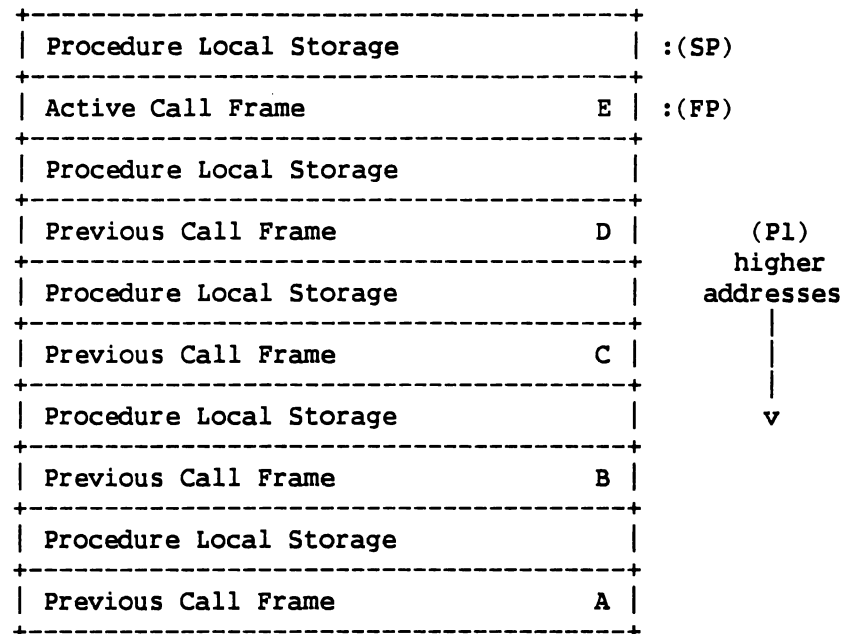
Above the call frame - that is in lower virtual addresses - is data local to the procedure concerned. The stack pointer points to the final address of the local storage section thus:



The whole data structure is called the procedure's Stack Frame

HANDLING EXCEPTIONS

Below is a schematic representation of a collection of stack frames after procedure A has called B, B has called C, C has called D and D has called E. E is the procedure executing



The important item so far as an exception handler is concerned is the first longword of the call frame - exception handler. We saw earlier that when an exception is raised the VAXELN kernel stops normal execution of the program temporarily and calls an exception handler. In addition it performs the following operations:

1. creates an argument list describing the exception
2. searches the current stack frames to find a call frame that contains a first longword that is non-zero
3. if 2. above is satisfied it invokes the handler at the address found stored in that longword

The general form of the argument list created for an exception handlers is:

```
Continue = Handler ( SIGNAL_ARGS,
                    MECHANISM_ARGS )
```

In VAXELN there is predeclared function defined thus:

```
FUNCTION EXCEPTION_HANDLER
( VAR SIGNAL_ARGS : CHF$R_SIGNAL_ARGS;
  VAR MECH_ARGS   : CHF$R_MECH_ARGS   ) : BOOLEAN; FUNCTION_TYPE
```

The type definitions CHF\$R_SIGNAL_ARGS and CHF\$R_MECH_ARGS are:

```
TYPE
  CHF$R_SIGNAL_ARGS = RECORD
    Arg_count      : INTEGER;
    Name           : INTEGER;
    Additional     : ARRAY [1..250] OF INTEGER;
  END;

  CHF$R_MECH_ARGS  = RECORD
    Arg_count      : INTEGER;
    Frame          : ^ANYTYPE;
    Depth          : INTEGER;
    SavR0          : INTEGER;
    SavR1          : INTEGER;
  END;
```

A further definition exists for handling additional arguments:

```
TYPE
  CHF$R_SIGNAL_ARGS_ADDITIONAL (Arg_count : INTEGER) = RECORD
    Arg_array : ARRAY [1..Arg_count - 3] OF INTEGER;
    PC       : INTEGER;
    PSL      : INTEGER;
  END;
```

This flexible type may be used to typecast the additional arguments with the number of additional arguments as its extent value.

HANDLING EXCEPTIONS

The schematic form of each argument list is shown below:

Signal arguments:

+-----+	
n = additional longwords	SIGARGS(1)
+-----+	
Exception value	SIGARGS(2)
+-----+	
Optional additional arguments making up one or more message sequences	
+-----+	
PC	SIGARGS(n)
+-----+	
PSL	SIGARGS(n+1)
+-----+	

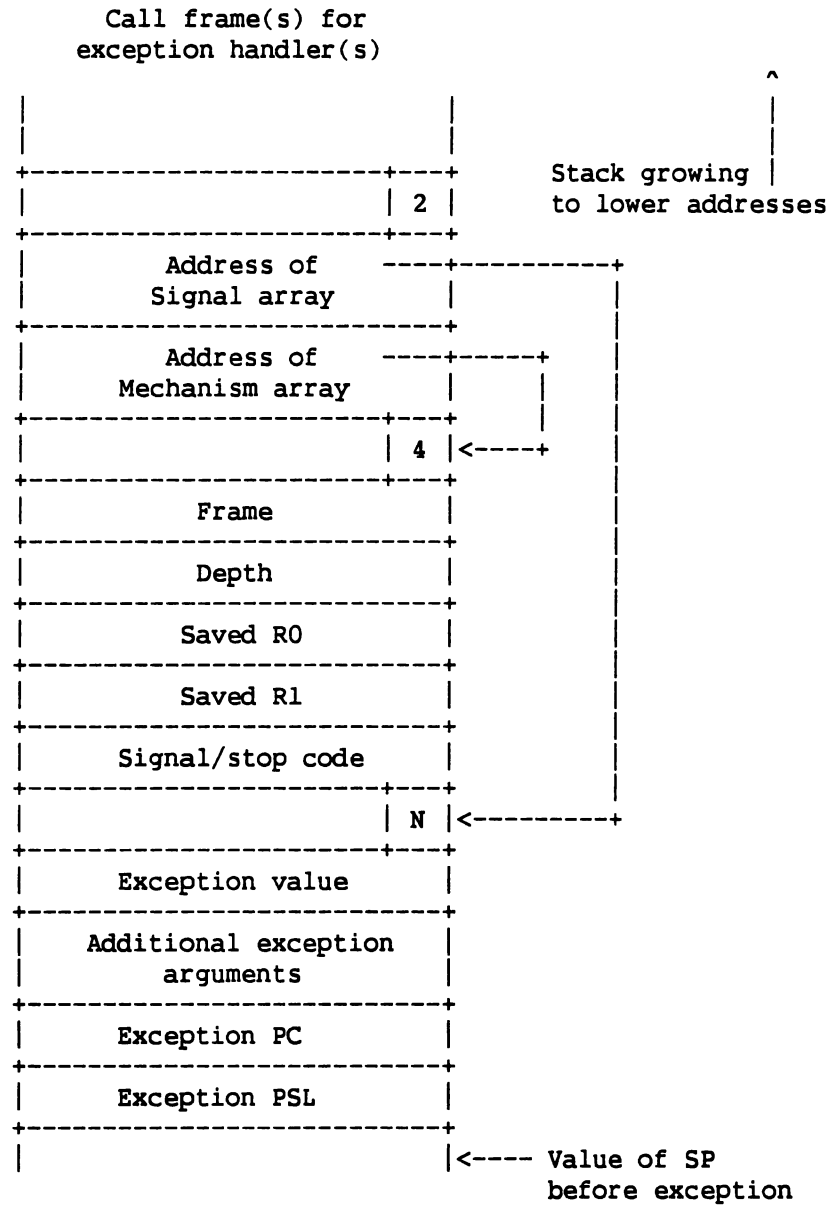
Mechanism arguments:

+-----+	
4 = additional longwords	MCHARGS(1)
+-----+	
Frame	MCHARGS(2)
+-----+	
Depth	MCHARGS(3)
+-----+	
Saved R0	MCHARGS(4)
+-----+	
Saved R1	MCHARGS(5)
+-----+	

(According to RTL Chapter 7)

15.9 STACK AFTER EXCEPTION OCCURS

The diagram shows the structures pushed onto the stack on the occurrence of an exception:



HANDLING EXCEPTIONS

15.10 POSSIBLE ACTION OF AN EXCEPTION HANDLER

An exception handler has three options:

1. discover which exception has occurred
2. decide whether to handle the exception or not to handle the exception
3. resume execution at an arbitrary point by unwinding some frames

15.10.1 Handling An Exception - CONTINUING

If the exception handler is able to take remedial action e.g. changing a zero denominator to a non-zero, then it will return a BOOLEAN value TRUE (remember: the exception handler function type is BOOLEAN). This course of action is termed CONTINUING (the kernel exception handling software receives a status code of SS\$_CONTINUE).

15.10.2 Not Handling An Exception - RESIGNALING

If the exception handler decides that it is unable to handle a particular condition it will return a BOOLEAN value FALSE. This is termed RESIGNALING (the kernel exception handling software receives a status code of SS\$_RESIGNAL).

15.10.3 Effect Of Resignalling

Resignalling - returning FALSE as the return value of an exception handler - causes the kernel to search the stack for another exception handler

15.10.4 Unwinding The Call Stack

Assume this calling sequence:

Procedure name	Calls	Establishes handler
A		AH
A	B	BH
B	C	CH
C	D	DH
D	E	EH

Procedure E raises an exception and its exception handler resignals as do exception handlers DH and CH.

Exception handler BH decides to UNWIND the call stack to the procedure that established it - B. The kernel routine KER\$UNWIND performs the following:

- (1) reviews the call stack for a handler
- (2) finds BH - UNWIND's own return PC is to BH
- (3) modifies return PC to be an address of a routine (on VMS it is called STARTUNWIND)
- (4) modifies saved PCs with another of its addresses (on VMS it is called LOOPUNWIND). Call frames for the procedures A, B, C, D and E lie beneath the Signal and Mechanism arguments on the stack and the frames for the exception handlers activated
- (5) if BH specified an alternate PC this is placed in the return PC of the frame above the one specified in the original call to unwind i.e. the frame for procedure C. (This enables a return from C to another location other than the instruction in B following the call to C).

Having modified all the frames unwinding can proceed as follows:

- (1) Unwind returns control to handler BH
- (2) BH processes the condition - the status returned by BH is not used by the unwinding routines
- (3) restores R0 and R1 from the mechanism array then
 - (a) if a handler exists for this frame calls it with signal name SS\$_UNWIND
 - (b) if R0 or R1 are in the register save mask it places the value of that register in the save area of the call frame
 - (c) returns control to the address in the saved PC
- (4) the call frame for that procedure is then discarded i.e. procedure E in our example
- (5) proceeds with steps (a), (b) and (c) above
- (6) the return from procedure C removes its call frame from the stack returning to the PC therein - either to the next instruction in B after the call to C or an alternate PC if requested in the earlier UNWIND

15.11 EXCEPTIONS DURING EXCEPTIONS

If an exception is raised while another exception is being processed the stack

HANDLING EXCEPTIONS

is searched for a handler that has not been tested already. This ensures that handlers are not invoked recursively

15.12 ROUTINES FOR EXCEPTION HANDLING

[LRM:13-7]

The usual VAX Pascal routines ESTABLISH and REVERT are available in VAXELN Pascal for establishing and removing an exception handler

15.13 MESSAGE FACILITIES

[RF:11-12]

Messages of the form:

```
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
\NOVERB\
```

may be generated from VAXELN. Indeed a useful addition to checking for even status codes is a routine to print explanatory messages. A procedure supplied to perform that function is ELN\$GET_STATUS_TEXT. Its use requires the inclusion of the module \$GET_MESSAGE_TEXT from RTLOBJECT.OLB.

The call format for this procedure looks like this:

```
ELN$GET_STATUS_TEXT (    msgid,
                        flags,
                        result_string
                        );
```

The flags argument is a SET of status fields permitting selection of one or more of the usual VMS message fields:

- o status\$facility
- o status\$severity
- o status\$ident
- o status\$text

A call to ELN\$GET_MESSAGE_TEXT with an empty set for the flags argument will provide all four fields of the message. Several of the programs associated with this course use a specially-written routine to print messages when EVEN status codes are reported. (see module CHECK_STATUS_AND_REPORT.PAS).

APPENDIX A
ASCII CHARACTER CODES

HEX	OCT	DEC	Char	Remarks
0	0	0	NUL	Null, tape feed, shift, ^P
1	1	1	SOH	Start of heading, start of message, ^A
2	2	2	STX	Start of text, end of address, ^B
3	3	3	ETX	End of text, end of message, ^C
4	4	4	EOT	End of transmission, ^D
5	5	5	ENQ	Enquiry, ^E
6	6	6	ACK	Acknowledge, ^F
7	7	7	BEL	Bell, ^G
8	10	8	BS	Backspace, format effector, ^H
9	11	9	HT	Horizontal tab, ^I
A	12	10	LF	Line feed, ^J
B	13	11	VT	Vertical tab, ^K
C	14	12	FF	Form feed, page, ^L
D	15	13	CR	Carriage return, ^M
E	16	14	SO	Shift out, ^N
F	17	15	SI	Shift in, ^O
10	20	16	DLE	Data link escape, ^P
11	21	17	DC1	Device control 1, ^Q
12	22	18	DC2	Device control 2, ^R
13	23	19	DC3	Device control 3, ^S
14	24	20	DC4	Device control 4, ^T
15	25	21	NAK	Negative acknowledge, ^U
16	26	22	SYN	Synchronous idle, ^V
17	27	23	ETB	End of transmission block, logical end of medium, ^W
18	30	24	CAN	Cancel, ^X
19	31	25	EM	End of medium, ^Y
1A	32	26	SUB	Substitute, ^Z
1B	33	27	ESC	Escape, prefix, shift, ^K
1C	34	28	FS	File separator, shift, ^L
1D	35	29	GS	Group separator, shift, ^M

ASCII CHARACTER CODES

HEX	OCT	DEC	Char	Remarks
1E	36	30	RS	Record separator, shift, ^N
1F	37	31	US	Unit separator, shift, ^O
20	40	32	SP	Space
21	41	33	!	Exclamation point
22	42	34	"	Double quotation mark
23	43	35	#	Number sign
24	44	36	\$	Dollar sign
25	45	37	%	Percent sign
26	46	38	&	Ampersand
27	47	39	'	Apostrophe
28	50	40	(Left parenthesis
29	51	41)	Right parenthesis
2A	52	42	*	Asterisk
2B	53	43	+	Plus sign
2C	54	44	,	Comma
2D	55	45	-	Minus sign
2E	56	46	.	Period, dot
2F	57	47	/	Slash, statement separator
30	60	48	0	Zero
31	61	49	1	One
32	62	50	2	Two
33	63	51	3	Three
34	64	52	4	Four
35	65	53	5	Five
36	66	54	6	Six
37	67	55	7	Seven
38	70	56	8	Eight
39	71	57	9	Nine
3A	72	58	:	Colon
3B	73	59	;	Semicolon
3C	74	60	<	Left angle bracket
3D	75	61	=	Equal sign
3E	76	62	>	Right angle bracket
3F	77	63	?	Question mark
40	100	64	@	At sign
41	101	65	A	Uppercase A
42	102	66	B	Uppercase B
43	103	67	C	Uppercase C
44	104	68	D	Uppercase D
45	105	69	E	Uppercase E

ASCII CHARACTER CODES

HEX	OCT	DEC	Char	Remarks
46	106	70	F	Uppercase F
47	107	71	G	Uppercase G
48	110	72	H	Uppercase H
49	111	73	I	Uppercase I
4A	112	74	J	Uppercase J
4B	113	75	K	Uppercase K
4C	114	76	L	Uppercase L
4D	115	77	M	Uppercase M
4E	116	78	N	Uppercase N
4F	117	79	O	Uppercase O
50	120	80	P	Uppercase P
51	121	81	Q	Uppercase Q
52	122	82	R	Uppercase R
53	123	83	S	Uppercase S
54	124	84	T	Uppercase T
55	125	85	U	Uppercase U
56	126	86	V	Uppercase V
57	127	87	W	Uppercase W
58	130	88	X	Uppercase X
59	131	89	Y	Uppercase Y
5A	132	90	Z	Uppercase Z
5B	133	91	[Left bracket, shift K
5C	134	92	\	Backslash, shift L
5D	135	93]	Right bracket, shift M
5E	136	94	^	Caret
5F	137	95	_	Underscore
60	140	96	`	Accent, grave
61	141	97	a	Lowercase a
62	142	98	b	Lowercase b
63	143	99	c	Lowercase c
64	144	100	d	Lowercase d
65	145	101	e	Lowercase e
66	146	102	f	Lowercase f
67	147	103	g	Lowercase g
68	150	104	h	Lowercase h
69	151	105	i	Lowercase i
6A	152	106	j	Lowercase j
6B	153	107	k	Lowercase k
6C	154	108	l	Lowercase l
6D	155	109	m	Lowercase m

ASCII CHARACTER CODES

HEX	OCT	DEC	Char	Remarks
6E	156	110	n	Lowercase n
6F	157	111	o	Lowercase o
70	160	112	p	Lowercase p
71	161	113	q	Lowercase q
72	162	114	r	Lowercase r
73	163	115	s	Lowercase s
74	164	116	t	Lowercase t
75	165	117	u	Lowercase u
76	166	118	v	Lowercase v
77	167	119	w	Lowercase w
78	170	120	x	Lowercase x
79	171	121	y	Lowercase y
7A	172	122	z	Lowercase z
7B	173	123	{	Left brace
7C	174	124		Vertical line
7D	175	125	}	Right brace
7E	176	126	~	Tilde
7F	177	127	DEL	Delete, rubout

APPENDIX B
EXAMPLES OF LANGUAGE EXTENSIONS

B.1 INTRODUCTION

This appendix contains examples of some of the facilities available in VAXELN Pascal.

The sources for the programs listed and command procedures to build the programs/systems are available from your instructor. Where appropriate, explanatory text precedes each example. Whenever possible line numbers have been inserted into the listings to facilitate discussion

B.2 EXAMPLE OF IDENT - SIMPLE.PAS

```
1  {-----
2  SOURCE:          SIMPLE.PAS
3
4  PURPOSE:         Demonstrates a simple MODULE and PROGRAM in VAXELN
5                    Pascal
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG SIMPLE
8
9  LINK:            $ LINK /DEBUG SIMPLE -
10                  _$ ,ELN$:RTLSHARE /LIBRARY -
11                  _$ ,RTL /LIBRARY
12
13 BUILD:           $ EBUILD /NOEDIT SIMPLE
14
15 NOTES:           1) Command procedure SIMPLE.COM compiles, links and
16                  builds this module into a VAXELN system or for
17                  running under VMS
18  -----}
```

EXAMPLES OF LANGUAGE EXTENSIONS

```
19
20 MODULE Simple [IDENT ('V1.000')];
21
22 { The IDENT field appears in the object file's MODULE HEADER with the
23   module's name thus:
24
25       1.  MODULE HEADER (OBJ$C_HDR_MHD), 53 bytes
26
27           structure level: 0
28           maximum record size: 512
29           module name: "SIMPLE"
30           module version: "V1.000"
31           creation   date/time: 13-FEB-1987 11:21
32
33       _____
34       To obtain this information proceed thus:
35           $ EPASCAL SIMPLE
36           $ ANALYZE /OBJECT SIMPLE
37   }
38 PROGRAM Simple (OUTPUT);
39
40     BEGIN
41
42         WRITELN ( 'A simple program to test VAXELN' );
43
44     END { of PROGRAM Simple }.
45 END { of  MODULE Simple };
```

B.2.1 Running SIMPLE

SIMPLE can be built using the command procedure of the same name:

This command procedure invokes other command procedures - all are available from your instructor

```

1      $      ! SIMPLE.COM
2      $      ! Command procedure to build the VAXELN module SIMPLE
3      $      !
4      $      ON ERROR THEN GOTO Switch_off_verify
5      $      SET DEFAULT Default_directory
6      $ !
7      $      INQUIRE -
8              /NOPUNCTUATION Running_under_VMS -
9              "Do you wish to run this program under VMS [Y/N]: "
10     $      IF .NOT. Running_under_VMS THEN GOTO Compile_simple
11     $ !
12     $      SET VERIFY
13     $      EPASCAL /LIST SIMPLE
14     $      LINK SIMPLE
15     $      GOTO Switch_off_verify
16     $ !
17     $ Compile_Simple:
18     $          @ELN_COMPILE_1  SIMPLE
19     $ Link_Simple:
20     $          @ELN_LINK_1     SIMPLE
21     $ System_build:
22     $          @ELN_EBUILD_1   SIMPLE
23     $ Switch_off_verify:
24     $      SET NOVERIFY
25     $      EXIT

```

The .DAT file used by EBUILD is listed below:

```

1      characteristic /noconsole
2      program SIMPLE /debug

```


EXAMPLES OF LANGUAGE EXTENSIONS

The command procedures invoked in the body of SIMPLE.COM are used by several of the command procedures on this course. They are listed below for convenience.

```
1 $      ! ELN_COMPILE_1.COM
2 $      ! Command procedure to compile a VAXELN Pascal program
3 $      !
4 $ Compile:
5 $      SET VERIFY
6 $      EPASCAL -
7          /LIST -
8          /DEBUG 'P1'
9 $      SET NOVERIFY
10 $     EXIT
```

```
1 $      ! ELN_LINK_1.COM
2 $      ! Command procedure to link VAXELN object modules
3 $      !
4 $ Link:
5 $      SET VERIFY
6 $      LINK -
7          /DEBUG 'P1' -
8          ,ELN$:RTLSHARE /LIBRARY -
9          ,ELN$:RTL /LIBRARY
10 $     SET NOVERIFY
11 $     EXIT
```

```
1
2 $      ! ELN_EBUILD_1.COM
3 $      ! Command procedure to build a VAXELN system from existing
4 $      ! 'P1'.DAT file
5 $      !
6 $ System_build:
7 $      SET VERIFY
8 $      EBUILD -
9          /NOEDIT 'P1'
10 $     SET NOVERIFY
11 $     EXIT
```

B.3 EXAMPLE OF %INCLUDE - INCLUDE_SOURCE.PAS

The %INCLUDE statement on line 32 of MODULE INCLUDE_SOURCE includes the text:

```
1      {-----}
2      SOURCE:      INCLUDE_SOURCE.INC
3
4      PURPOSE:      For inclusion in INCLUDE_SOURCE.PAS
5
6      USED BY:      INCLUDE_SOURCE.PAS
7      -----}
8      PROCEDURE Show_scope;
9
10     BEGIN
11         WRITELN ( 'This is from procedure SHOW_SCOPE' );
12         WRITELN ( ' about to write GREETING.....' );
13         WRITELN ( Greeting );
14
15     END { PROCEDURE Show_scope };
```

EXAMPLES OF LANGUAGE EXTENSIONS

The program looks like this:

```
1      {-----}
2      SOURCE:      INCLUDE_SOURCE.PAS
3
4      PURPOSE:      Demonstrates a simple MODULE and PROGRAM in VAXELN
5                     Pascal with outer level declaration of a CONST
6
7      NEEDS:        1) INCLUDE_SOURCE.INC for the %INCLUDE
8
9      COMPILE:      $ EPASCAL /LIST /DEBUG INCLUDE_SOURCE
10
11     LINK:         $ LINK /DEBUG INCLUDE_SOURCE, ELN$:RTLSHARE /LIBRARY, -
12                   _$ RTL /LIBRARY
13
14     BUILD:        $ EBUILD /NOEDIT INCLUDE_SOURCE
15
16     NOTES:        1) Command procedure INCLUDE_SOURCE.COM compiles,
17                   links and builds this module into a VAXELN system
18                   or for running under VMS
19     -----}
20
21     MODULE INCLUDE_SOURCE [IDENT ('V1.000')];
22
23         CONST     { This outer level declaration is EXPORTed explicitly }
24                   Greeting = 'A simple program to test VAXELN';
25
26     PROGRAM Simple (OUTPUT);
27
28         CONST
29             Greeting = 'This is an inner declaration of the same name';
30
31     { Code for the procedure SHOW_SCOPE }
32         %INCLUDE 'INCLUDE_SOURCE.INC/LIST'
33
34         BEGIN
35
36             WRITELN ( Greeting );
37             { prints "This is an inner declaration of the same name" }
38
39             Show_scope;
40
41         END { of PROGRAM }.
42     END      { of MODULE };
```

B.3.1 Running INCLUDE_SOURCE

INCLUDE_SOURCE can be built using the command procedure of the same name:

```

1      $      ! INCLUDE_SOURCE.COM
2      $      ! Command procedure to build the VAXELN
3      $      ! module INCLUDE_SOURCE
4      $      !
5      $      ON ERROR THEN GOTO Switch_off_verify
6      $      SET DEFAULT Default_directory
7      $ !
8      $      INQUIRE -
9              /NOPUNCTUATION Running_under_VMS -
10             "Do you wish to run this program under VMS [Y/N]: "
11      $      IF .NOT. Running_under_VMS THEN GOTO Compile
12      $ !
13      $      SET VERIFY
14      $      EPASCAL /LIST INCLUDE_SOURCE
15      $      LINK      INCLUDE_SOURCE
16      $      GOTO Switch_off_verify
17      $ !
18      $ Compile:
19      $              @ELN_COMPILE_1  INCLUDE_SOURCE
20      $ Link:
21      $              @ELN_LINK_1     INCLUDE_SOURCE
22      $ System_build:
23      $              @ELN_EBUILD_1   INCLUDE_SOURCE
24      $ Switch_off_verify:
25      $      SET NOVERIFY
26      $      EXIT

```

The .DAT file used by EBUILD is listed below:

```

1      characteristic /noconsole
2      program INCLUDE_SOURCE /debug

```

When INCLUDE_SOURCE is run the output looks like this:

```

$ RUN INCLUDE_SOURCE
This is an inner declaration of the same name
This is from procedure SHOW_SCOPE
  about to write GREETING.....
This is an inner declaration of the same name

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.4 EXAMPLE OF IMPORT AND EXPORT - IMPORT_1.PAS

```
1  {-----}
2  SOURCE:      IMPORT_1.PAS
3
4  PURPOSE:     This module converts a temperature in CELSIUS to
5                FAHRENHEIT for user input from a terminal.
6                It demonstrates the IMPORT statement in VAXELN
7                Pascal
8
9  COMPILE:     $ EPASCAL /DEBUG IMPORT_1, -
10              _$ EXPORT_1/MODULE
11
12  LINK:       $ LINK      /DEBUG IMPORT_1, EXPORT_1 -
13              _$ ,ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY
14
15  BUILD:      $ EBUILD /NOEDIT IMPORT_1
16
17  NOTES:      1) Command procedure IMPORT_1.COM compiles, links and
18                builds this module into a VAXELN system or for
19                running under VMS
20  {-----}
21  MODULE Temperature_converter [IDENT ('V1.000')];
22
23  IMPORT  Clear_screen,
24          Prompt,
25          Celsius_input,
26          Fahrenheit_value,
27          Conversion_factor,
28          Fahrenheit_zero;
29
30  PROGRAM Convert_C_to_F ( INPUT, OUTPUT );
31
32  VAR
33      Celsius,
34      Fahrenheit      : REAL;
35
36  { ----- }
37
38  PROCEDURE Describe_program;
39
40  CONST
41      LF = '(10);    { Line feed }
42
43  BEGIN
44      WRITELN ( Clear_screen );
45      WRITELN ( 'This program calculates a temperature in',
46                ' FAHRENHEIT from a' );
47      WRITELN ( 'value in CELSIUS input by the user', LF, LF );
48      WRITELN ( 'You will be prompted for input', LF, LF );
49  END;
50
```

```

51 { ----- }
52
53 PROCEDURE Prompt_user;
54
55     BEGIN
56         WRITE ( Prompt );
57     END;
58
59 { ----- }
60
61 FUNCTION Read_Celsius_value : REAL;
62
63     VAR
64         Temporary : REAL;
65
66     BEGIN
67         READ ( Temporary );  READLN;
68         Read_Celsius_value := Temporary;
69     END;
70
71 { ----- }
72
73 FUNCTION Calculate_Fahrenheit ( C_value : REAL ) : REAL ;
74
75     BEGIN
76         Calculate_Fahrenheit := C_value * Conversion_factor +
77                                 Fahrenheit_zero;
78     END;
79
80 { ----- }
81
82 PROCEDURE Write_results ( C, F : REAL );
83
84     BEGIN
85         WRITELN ( Celsius_input,   C :1:1,
86                 Fahrenheit_value, F :1:1 );
87     END;
88
89 { ----- }
90
91 BEGIN { Body of program }
92
93     Describe_program;
94     Prompt_user;
95     Celsius := Read_Celsius_value;
96     Fahrenheit := Calculate_Fahrenheit ( Celsius );
97     Write_results ( Celsius, Fahrenheit );
98 END.
99
100 END { of module };

```

EXAMPLES OF LANGUAGE EXTENSIONS

```

1  {-----}
2  SOURCE:      EXPORT_1.PAS
3
4  COMPILE:      $ EPASCAL EXPORT_1
5
6  USED BY:      IMPORT_1.PAS
7  {-----}
8
9  MODULE EXPORT_1 [IDENT ('V1.000')];
10
11 { Outer level declarations }
12 { The exporting is implied in the outer level declarations
13   of a module }
14
15 EXPORT
16     Clear_screen,
17     Prompt,
18     Celsius_input,
19     Fahrenheit_value,
20     Conversion_factor,
21     Fahrenheit_zero;
22
23 CONST
24     Clear_screen      = '(27)'[2J'(27)'][H';
25                       { translates to <ESC>[2J<ESC>[H }
26     Prompt            = 'Please enter a temperature in Celsius: ';
27     Celsius_input     = 'The Celsius temperature input was ';
28     Fahrenheit_value  = ' and its Fahrenheit equivalent is ';
29     Conversion_factor = 1.8; { result of 9.0 / 5.0 }
30     Fahrenheit_zero   = 32.0;
31
32 END.

```

B.4.1 Running IMPORT_1

IMPORT_1 can be built using the command procedure of the same name:

```

1      $      ! IMPORT_1.COM
2      $      ! Command procedure to build the VAXELN module
3      $      !      IMPORT_1
4      $      !
5      $      ON ERROR THEN GOTO Switch_off_verify
6      $      SET DEFAULT Default_directory
7      $ !
8      $      INQUIRE -
9              /NOPUNCTUATION Running_under_VMS -
10             "Do you wish to run this program under VMS [Y/N]: "
11      $      IF .NOT. Running_under_VMS THEN GOTO Compile_exporter
12      $ !
13      $      SET VERIFY
14      $      EPASCAL /LIST EXPORT_1
15      $      EPASCAL /LIST IMPORT_1, EXPORT_1 /MODULE
16      $      LINK IMPORT_1, EXPORT_1
17      $      GOTO Switch_off_verify
18      $ !
19      $ Compile_exporter:
20      $      SET VERIFY
21      $      EPASCAL -
22              /LIST -
23              EXPORT_1      ! contains exported symbols for import
24                          by IMPORT_1
25      $ Compile_importer:
26      $      EPASCAL -
27              /LIST -
28              /DEBUG IMPORT_1 -
29              ,EXPORT_1 /MODULE
30      $ Link_importer_exporter:
31      $      LINK -
32              /DEBUG IMPORT_1 -
33              ,EXPORT_1 -
34              ,ELN$:RTLSHARE /LIBRARY -
35              ,ELN$:RTL /LIBRARY
36      $ System_build:
37      $      EBUILD -
38                      /NOEDIT IMPORT_1
39      $ Switch_off_verify:
40      $      SET NOVERIFY
41      $      EXIT

```


EXAMPLES OF LANGUAGE EXTENSIONS

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program IMPORT_1 /debug
```

When IMPORT_1 is run the output looks like this:

```
$ RUN IMPORT_1
This program calculates a temperature in FAHRENHEIT from a
value in CELSIUS input by the user
```

You will be prompted for input

Please enter a temperature in Celsius: 13.2

The Celsius temperature input was 13.2 and its Fahrenheit equivalent is 55.8

B.5 EXAMPLE OF INCLUDE - INCLUDE_1.PAS

```

1  {-----}
2  SOURCE:      INCLUDE_1.PAS
3
4  PURPOSE:     This module converts a temperature in CELSIUS to
5                FAHRENHEIT for user input from a terminal.
6                It demonstrates the INCLUDE statement in VAXELN
7                Pascal
8
9  COMPILE:     $ EPASCAL /DEBUG INCLUDE_1, VAXELN-MODULES /LIBRARY
10
11 LINK:        $ LINK      /DEBUG INCLUDE_1, VAXELN-MODULES /LIBRARY -
12              _$ ,ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY
13
14 BUILD:       $ EBUILD /NOEDIT INCLUDE_1
15
16 NOTES:       1) Command procedure INCLUDE_1.COM compiles, links and
17                builds this module into a VAXELN system or for
18                running under VMS
19 -----}
20 MODULE Temperature_converter [IDENT ('V1.000')];
21
22 INCLUDE INCLUDE_1_DEFS;    { In the object library: VAXELN-MODULES }
23
24 PROGRAM Convert_C_to_F ( INPUT, OUTPUT );
25
26     VAR
27         Celsius,
28         Fahrenheit      : REAL;
29
30 { ----- }
31
32 PROCEDURE Describe_program;
33
34     CONST
35         LF = '(10);    { Line feed }
36
37     BEGIN
38         WRITELN ( Clear_screen );
39         WRITELN ( 'This program calculates a temperature in',
40                 ' FAHRENHEIT from a' );
41         WRITELN ( 'value in CELSIUS input by the user', LF, LF );
42         WRITELN ( 'You will be prompted for input', LF, LF );
43     END;
44

```

EXAMPLES OF LANGUAGE EXTENSIONS

```

45 { ----- }
46
47 PROCEDURE Prompt_user;
48
49     BEGIN
50         WRITE ( Prompt );
51     END;
52
53 { ----- }
54
55 FUNCTION Read_Celsius_value : REAL;
56
57     VAR
58         Temporary : REAL;
59
60     BEGIN
61         READ ( Temporary );  READLN;
62         Read_Celsius_value := Temporary;
63     END;
64
65 { ----- }
66
67 FUNCTION Calculate_Fahrenheit ( C_value : REAL ) : REAL ;
68
69     BEGIN
70         Calculate_Fahrenheit := C_value * Conversion_factor +
71                                 Fahrenheit_zero;
72     END;
73
74 { ----- }
75
76 PROCEDURE Write_results ( C, F : REAL );
77
78     BEGIN
79         WRITELN ( Celsius_input,    C :1:1,
80                  Fahrenheit_value, F :1:1 );
81     END;
82
83 { ----- }
84
85 BEGIN { Body of program }
86
87     Describe_program;
88     Prompt_user;
89     Celsius      := Read_Celsius_value;
90     Fahrenheit := Calculate_Fahrenheit ( Celsius );
91     Write_results ( Celsius, Fahrenheit );
92 END.
93
94 END { of module Temperature converter };

```

The file INCLUDE_1_DEFS.PAS is listed below:

```

1  {-----}
2  SOURCE:      INCLUDE_1_DEFS.PAS
3
4  COMPILE:      $ EPASCAL /DEBUG INCLUDE_1_DEFS
5
6  USED BY:      INCLUDE_1.PAS
7  -----}
8
9  MODULE INCLUDE_1_DEFS [IDENT ('V1.000')];
10
11 { Outer level declarations }
12 { The exporting is implied in the outer level
13   declarations of a module }
14
15 EXPORT
16     Clear_screen,
17     Prompt,
18     Celsius_input,
19     Fahrenheit_value,
20     Conversion_factor,
21     Fahrenheit_zero;
22
23 CONST
24     Clear_screen      = '(27)'[2J'(27)'[H';
25                       { translates to <ESC>[2J<ESC>[H }
26     Prompt            = 'Please enter a temperature in Celsius: ';
27     Celsius_input     = 'The Celsius temperature input was ';
28     Fahrenheit_value  = ' and its Fahrenheit equivalent is ';
29     Conversion_factor = 1.8; { result of 9.0 / 5.0 }
30     Fahrenheit_zero   = 32.0;
31
32 END.

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.5.1 Running INCLUDE_1

INCLUDE_1 can be built using the command procedure of the same name:

```
1 $      ! INCLUDE_1.COM
2 $      ! Command procedure to build the VAXELN module
3 $      ! INCLUDE_1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile_Includer
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST INCLUDE_1, VAXELN-MODULES /LIBRARY
15 $      LINK INCLUDE_1, VAXELN-MODULES /LIBRARY
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile_Includer:
19 $      SET VERIFY
20 $      EPASCAL -
21          /LIST -
22          /DEBUG INCLUDE_1 -
23          ,VAXELN-MODULES/LIBRARY
24 $ Link_Includer:
25 $      LINK -
26          /DEBUG INCLUDE_1 -
27          ,VAXELN-MODULES /LIBRARY -
28          ,ELN$:RTLSHARE /LIBRARY -
29          ,ELN$:RTL /LIBRARY
30 $ System_build:
31 $      EBUILD -
32          /NOEDIT INCLUDE_1
33 $ Switch_off_verify:
34 $      SET NOVERIFY
35 $      EXIT
```

EXAMPLES OF LANGUAGE EXTENSIONS

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program INCLUDE_1 /debug
```

When INCLUDE_1 is run the output looks like this:

```
$ RUN IMPORT_1
This program calculates a temperature in FAHRENHEIT from a
value in CELSIUS input by the user
```

You will be prompted for input

Please enter a temperature in Celsius: 23.5

The Celsius temperature input was 23.5 and its Fahrenheit equivalent is 74.3

EXAMPLES OF LANGUAGE EXTENSIONS

B.6 EXAMPLE OF RADIX SPECIFIERS - RADIX_1.PAS

```
1  {-----}
2  SOURCE:          RADIX_1.PAS
3
4  PURPOSE:         Demonstrates the following facilities:
5
6                    1) Radix specifiers
7                    2) BIN function
8                    3) HEX function
9                    4) OCT function
10
11  COMPILE:         $ EPASCAL /LIST /DEBUG RADIX_1
12
13  LINK:            $ LINK /DEBUG RADIX_1 -
14                  _$ ,ELN$:RTLSHARE /LIBRARY -
15                  _$ ,RTL /LIBRARY
16
17  BUILD:           $ EBUILD /NOEDIT RADIX_1
18
19  NOTES:           1) Command procedure RADIX_1.COM compiles, links and
20                  builds this module into a VAXELN system or for
21                  running under VMS
22  -----}
23  MODULE RADIX_1;
24
25  CONST
26      Binary_value   = %B'01010101 10101010 01010101 10101010';
27      Hexadecimal_value = %X'ABCD DCBA';
28      Octal_value     = %O177777;
29
30  PROGRAM RADIX_1 ( OUTPUT );
31
32  BEGIN
33      WRITELN ( 'Binary constant is: ', Binary_value );
34      WRITELN ( 'Hex      constant is: ', Hexadecimal_value );
35      WRITELN ( 'Octal   constant is: ', Octal_value );
36
37      WRITELN;
38      WRITELN ( 'Now with the BIN HEX and OCT functions ... ' );
39      WRITELN;
40      WRITELN ( 'Binary constant is: ', BIN (Binary_value) );
41      WRITELN ( 'Hex      constant is: ', HEX (Hexadecimal_value) );
42      WRITELN ( 'Octal   constant is: ', OCT (Octal_value) );
43
44  END { of PROGRAM RADIX_1 }.
45  END { of MODULE RADIX_1 };
```

B.6.1 Running RADIX_1

RADIX_1 can be built using the command procedure of the same name:

```

1 $      ! RADIX_1.COM
2 $      ! Command procedure to build the VAXELN module
3 $      ! RADIX_1
4 $ !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST RADIX_1
15 $      LINK RADIX_1
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile:
19 $          @ELN_COMPILE_1 RADIX_1
20 $ Link:
21 $          @ELN_LINK_1 RADIX_1
22 $ System_build:
23 $          @ELN_EBUILD_1 RADIX_1
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT

```

The .DAT file used by EBUILD is listed below:

```

1      characteristic /noconsole
2      program RADIX_1 /debug

```

When RADIX_1 is run the output looks like this:

```

$ RUN RADIX_1
Binary constant is: 1437226410
Hex      constant is: -1412571974
Octal    constant is:      65535

Now with the BIN HEX and OCT functions ...

Binary constant is: 01010101101010100101010110101010
Hex      constant is: ABCDDCBA
Octal    constant is: 00000177777

```


EXAMPLES OF LANGUAGE EXTENSIONS

B.7 EXAMPLE OF SIZE ATTRIBUTES - ATTRIBS_1.PAS

```
1  {-----}
2  SOURCE:      ATTRIBS_1.PAS
3
4  PURPOSE:      Demonstrates the following attributes and facilities:
5
6                  1) POS attribute
7                  2) BIT attribute
8                  3) Extent expression with 1) and 2) above
9                  4) WORD attribute
10                 5) Typecasting
11                 6) BIN function
12
13  COMPILE:      $ EPASCAL /LIST /DEBUG ATTRIBS_1
14
15  LINK:         $ LINK /DEBUG ATTRIBS_1 -
16                _$ ,ELN$:RTLSHARE /LIBRARY -
17                _$ ,RTL /LIBRARY
18
19  BUILD:        $ EBUILD /NOEDIT ATTRIBS_1
20
21  NOTES:        1) Command procedure ATTRIBS_1.COM compiles, links and
22                builds this module into a VAXELN system or for
23                running under VMS
24  {-----}
25  MODULE ATTRIBS_1;
26
27  CONST
28      Pair      = 2;    { for use as extent expression }
29      Trio      = 3;    { for use as extent expression }
30
31  PROGRAM ATTRIBS_1 ( OUTPUT );
32
33  TYPE
34      Word       = [WORD] -32768..32767;
35
36  VAR
37      Bit_pattern : [WORD] PACKED RECORD
38          L_S_Bit  : [BIT]          0..1; { least sig. bit }
39          Bits_02   : [POS (2), BIT (Trio)] 0..7;
40          Bits_08   : [POS (8), BIT (Pair)] 0..3;
41          M_S_Bit   : [POS(15), BIT]      0..1; { most sig. bit }
42      END;
43
```

```

44 BEGIN
45     WITH Bit_pattern DO
46     BEGIN
47         L_S_Bit := 1;
48         Bits_02 := 5;
49         Bits_08 := 3;
50         M_S_Bit := 1;
51
52 {   After those assignments Bit_pattern looks like this:
53
54     +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
55     |15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|   Bit position
56     +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
57     | 1| 0| 0| 0| 0| 0| 0| 1| 1| 0| 0| 0| 1| 0| 1| 0| 1|   After setting
58     +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
59     |--|           |----|           |-----|   |--|
60     ^               ^               ^             ^
61   M_S_Bit           Bits_08           Bits_02     L_S_Bit
62
63 }
64     WRITELN ( ' L_S_Bit: ', L_S_Bit:1 );
65     WRITELN ( ' Bits_02: ', Bits_02:1 );
66     WRITELN ( ' Bits_08: ', Bits_08:1 );
67     WRITELN ( ' M_S_Bit: ', M_S_Bit:1 );
68
69     WRITELN ( 'Binary pattern for RECORD is: ',
70               BIN (Bit_pattern::BYTE_DATA)      );
71     WRITELN ( 'Numeric value is: ', Bit_pattern::Word );
72     END;
73
74 END { of PROGRAM ATTRIBS_1 }.
75 END { of MODULE ATTRIBS_1 };

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.7.1 Running ATTRIBS_1

ATTRIBS_1 can be built using the command procedure of the same name:

```
1 $      ! ATTRIBS_1.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module ATTRIBS_1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST ATTRIBS_1
15 $      LINK ATTRIBS_1
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile:
19 $          @ELN_COMPILE_1  ATTRIBS_1
20 $ Link:
21 $          @ELN_LINK_1    ATTRIBS_1
22 $ System_build:
23 $          @ELN_EBUILD_1  ATTRIBS_1
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT
```

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program ATTRIBS_1 /debug
```

When ATTRIBS_1 is run the output looks like this:

```
$ RUN ATTRIBS_1
L_S_Bit: 1
Bits_02: 5
Bits_08: 3
M_S_Bit: 1
Binary pattern for RECORD is: 1000001100010101
Numeric value is:      -31979
```

B.8 EXAMPLE OF TYPECASTING - TYPECAST_1.PAS

```

1  {-----}
2  SOURCE:          TYPECAST_1.PAS
3
4  PURPOSE:         Demonstrates the following facilities:
5
6                      1) Typecasting
7                      2) BIN function
8                      3) Size attributes
9
10 COMPILE:         $ EPASCAL /LIST /DEBUG TYPECAST_1
11
12 LINK:            $ LINK /DEBUG TYPECAST_1 -
13                  _$ ,ELN$:RTLSHARE /LIBRARY -
14                  _$ ,RTL /LIBRARY
15
16 BUILD:           $ EBUILD /NOEDIT TYPECAST_1
17
18 NOTES:           1) Command procedure TYPECAST_1.COM compiles, links and
19                  builds this module into a VAXELN system or for
20                  running under VMS
21 -----}
22 MODULE Typecaster [IDENT('V1.000')];
23
24 PROGRAM Typecaster ( INPUT, OUTPUT );
25
26 TYPE
27     Byte_record = [BYTE] PACKED RECORD
28                   B1 : [POS(0), BIT(5)] 0..31;
29                   UC : [BIT(1)]         0..1;
30                   B3 : [BIT(2)]         0..3;
31     END;
32
33 {
34     Byte_record
35     +---+---+---+---+---+---+
36     | 7| 6| 5| 4| 3| 2| 1| 0|   Bit position
37     +---+---+---+---+---+---+
38     |-B3--|UC|-----B1-----|
39 }
40
41 VAR
42     Alphabetic      : BOOLEAN;
43     Bit_pattern     : Byte_record;
44     User_char       : CHAR;
45 BEGIN
46
47     WRITE ( 'Please enter character: ' );
48     READLN ( User_char );
49

```

EXAMPLES OF LANGUAGE EXTENSIONS

```
50 { Check if its in the alphabets }
51     Alphabetic := User_char IN [ 'A'..'Z', 'a'..'z' ];
52
53 { Typecast it to the byte record }
54     Bit_pattern::BYTE_DATA := User_char::BYTE_DATA;
55
56     WRITELN ( 'Bit pattern before: ', BIN (User_char::BYTE_DATA) );
57
58 { If it IS lowercase alphabetic bit 5 will be set.
59   Clear bit 5. That subtracts 32 from the byte value }
60
61     IF Alphabetic THEN
62         IF ODD      ( Bit_pattern.UC )
63             THEN BEGIN
64                 WRITELN ( 'Lowercase character' );
65                 Bit_pattern.UC := 0;
66                 User_char::BYTE_DATA := Bit_pattern::BYTE_DATA;
67                 WRITELN ( User_char, ' now uppercase' );
68             END
69
70         ELSE BEGIN
71             WRITELN ( 'Uppercase character' );
72         END;
73
74     WRITELN ( 'Bit pattern after : ', BIN (User_char::BYTE_DATA) );
75
76 END { of PROGRAM Typecaster }.
77 END { of MODULE Typecaster };
```

B.8.1 Running TYPECAST_1

TYPECAST_1 can be built using the command procedure of the same name:

```

1 $      ! TYPECAST_1.COM
2 $      ! Command procedure to build the VAXELN module TYPECAST_1
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $      IF .NOT. Running_under_VMS THEN GOTO Compile
11 $ !
12 $      SET VERIFY
13 $      EPASCAL /LIST TYPECAST_1
14 $      LINK TYPECAST_1
15 $      GOTO Switch_off_verify
16 $ !
17 $ Compile:
18 $          @ELN_COMPILE_1 TYPECAST_1
19 $ Link:
20 $          @ELN_LINK_1 TYPECAST_1
21 $ System_build:
22 $          @ELN_EBUILD_1 TYPECAST_1
23 $ Switch_off_verify:
24 $      SET NOVERIFY
25 $      EXIT

```

The .DAT file used by EBUILD is listed below:

```

1      characteristic /noconsole
2      program TYPECAST_1 /debug

```

EXAMPLES OF LANGUAGE EXTENSIONS

When TYPECAST_1 is run the output looks like this:

```
$ RUN TYPECAST_1
Please enter character: K
Bit pattern before: 01001011
Uppercase character
Bit pattern after : 01001011
```

```
$ RUN TYPECAST_1
Please enter character: s
Bit pattern before: 01110011
Lowercase character
S now uppercase
Bit pattern after : 01010011
```

```
$ RUN TYPECAST_1
Please enter character: &
Bit pattern before: 00100110
Bit pattern after : 00100110
```

B.9 EXAMPLE OF SCOPE - SCOPE_1.PAS

```

1  {-----}
2  SOURCE:      SCOPE_1.PAS
3
4  PURPOSE:     Demonstrates a simple MODULE and PROGRAM in VAXELN
5                Pascal with outer level declaration of a CONST
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG SCOPE_1
8
9  LINK:        $ LINK /DEBUG SCOPE_1, ELN$:RTLSHARE /LIBRARY, -
10              _$ RTL /LIBRARY
11
12 BUILD:       $ EBUILD /NOEDIT SCOPE_1
13
14 NOTES:       1) Command procedure SCOPE_1.COM compiles, links and
15                builds this module into a VAXELN system or for
16                running under VMS
17 -----}
18 MODULE SCOPE_1 [IDENT ('V1.000')];
19
20     CONST      { This outer level declaration is EXPORTed explicitly }
21         Greeting = 'A simple program to test VAXELN';
22
23 PROGRAM SCOPE_1 (OUTPUT);
24
25     BEGIN
26         WRITELN ( Greeting );
27
28 END { of PROGRAM SCOPE_1 }.
29
30 END { of MODULE  SCOPE_1 };

```


EXAMPLES OF LANGUAGE EXTENSIONS

B.9.1 Running SCOPE_1

SCOPE_1 can be built using the command procedure of the same name:

```
1 $      ! SCOPE_1.COM
2 $      ! Command procedure to build the VAXELN module SCOPE_1
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION  Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $      IF .NOT. Running_under_VMS THEN GOTO Compile
11 $ !
12 $      SET VERIFY
13 $      EPASCAL /LIST SCOPE_1
14 $      LINK SCOPE_1
15 $      GOTO Switch_off_verify
16 $ !
17 $ Compile:
18 $          @ELN_COMPILE_1  SCOPE_1
19 $ Link:
20 $          @ELN_LINK_1     SCOPE_1
21 $ System_build:
22 $          @ELN_EBUILD_1   SCOPE_1
23 $ Switch_off_verify:
24 $      SET NOVERIFY
25 $      EXIT
```

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program SCOPE_1 /debug
```

When SCOPE_1 is run the output looks like this:

```
$ RUN SCOPE_1
A simple program to test VAXELN
```

B.10 EXAMPLE OF SCOPE - SCOPE_2.PAS

```

1  {-----}
2  SOURCE:      SCOPE_2.PAS
3
4  PURPOSE:     Demonstrates a simple MODULE and PROGRAM in VAXELN
5                Pascal with outer level declaration of a CONST
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG SCOPE_2
8
9  LINK:        $ LINK /DEBUG SCOPE_2, ELN$:RTLSHARE /LIBRARY, -
10              _$ RTL /LIBRARY
11
12 BUILD:       $ EBUILD /NOEDIT SCOPE_2
13
14 NOTES:       1) Command procedure SCOPE_2.COM compiles, links and
15                builds this module into a VAXELN system or for
16                running under VMS
17  {-----}
18 MODULE SCOPE_2 [IDENT ('V1.000')];
19
20   CONST      { This outer level declaration is EXPORTed explicitly }
21             Greeting = 'A simple program to test VAXELN';
22
23 PROGRAM SCOPE_2 (OUTPUT);
24
25   CONST
26       Greeting = 'This is an inner declaration of the same name';
27
28   BEGIN
29
30       WRITELN ( Greeting );
31       { prints "This is an inner declaration of the same name" }
32
33 END { of PROGRAM SCOPE_2 }.
34 END { of MODULE SCOPE_2 };

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.10.1 Running SCOPE_2

SCOPE_2 can be built using the command procedure of the same name:

```
1 $      ! SCOPE_2.COM
2 $      ! Command procedure to build the VAXELN module SCOPE_2
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION  Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $     IF .NOT. Running_under_VMS THEN GOTO Compile
11 $ !
12 $     SET VERIFY
13 $     EPASCAL /LIST SCOPE_2
14 $     LINK SCOPE_2
15 $     GOTO Switch_off_verify
16 $ !
17 $ Compile:
18 $           @ELN_COMPILE_1  SCOPE_2
19 $ Link:
20 $           @ELN_LINK_1     SCOPE_2
21 $ System_build:
22 $           @ELN_EBUILD_1   SCOPE_2
23 $ Switch_off_verify:
24 $     SET NOVERIFY
25 $     EXIT
```

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program SCOPE_2 /debug
```

When SCOPE_2 is run the output looks like this:

```
$ RUN SCOPE_2
This is an inner declaration of the same name
```

B.11 EXAMPLE OF SCOPE - SCOPE_3.PAS

```

1  {-----}
2  SOURCE:      SCOPE_3.PAS
3
4  PURPOSE:     Demonstrates a simple MODULE and PROGRAM in VAXELN
5                Pascal with outer level declaration of a CONST and
6                PROCEDURE
7
8  COMPILE:     $ EPASCAL /LIST /DEBUG SCOPE_3
9
10 LINK:        $ LINK /DEBUG SCOPE_3, ELN$:RTLSHARE /LIBRARY, -
11              _$ RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT SCOPE_3
14
15 NOTES:       1) Command procedure SCOPE_3.COM compiles, links and
16                builds this module into a VAXELN system or for
17                running under VMS
18 {-----}
19 MODULE SCOPE_3 [IDENT ('V1.000')];
20
21     CONST     { This outer level declaration is EXPORTed explicitly }
22               Greeting = 'A simple program to test VAXELN';
23
24 {-----}
25 PROCEDURE Show_scope;
26
27     BEGIN
28         WRITELN ( 'This is from procedure SHOW_SCOPE about ',
29                 'to write GREETING' );
30         WRITELN ( Greeting );
31     END { PROCEDURE Show_scope };
32 {-----}
33
34 PROGRAM SCOPE_3 (OUTPUT);
35
36     CONST
37         Greeting = 'This is an inner declaration of the same name';
38
39     BEGIN
40
41         WRITELN ( Greeting );
42         { prints "This is an inner declaration of the same name" }
43         Show_scope;
44
45     END { of PROGRAM SCOPE_3 }.
46 END { of MODULE SCOPE_3 };

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.11.1 Running SCOPE_3

SCOPE_3 can be built using the command procedure of the same name:

```
1 $      ! SCOPE_3.COM
2 $      ! Command procedure to build the VAXELN module SCOPE_3
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $     IF .NOT. Running_under_VMS THEN GOTO Compile
11 $ !
12 $     SET VERIFY
13 $     EPASCAL /LIST SCOPE_3
14 $     LINK SCOPE_3
15 $     GOTO Switch_off_verify
16 $ !
17 $ Compile:
18 $           @ELN_COMPILE_1  SCOPE_3
19 $ Link:
20 $           @ELN_LINK_1     SCOPE_3
21 $ System_build:
22 $           @ELN_EBUILD_1   SCOPE_3
23 $ Switch_off_verify:
24 $     SET NOVERIFY
25 $     EXIT
```

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program SCOPE_3 /debug
```

When SCOPE_3 is run the output looks like this:

```
$ RUN SCOPE_3
This is an inner declaration of the same name
This is from procedure SHOW_SCOPE about to write GREETING
A simple program to test VAXELN
```

B.12 EXAMPLE OF SCOPE - SCOPE_3A.PAS

```

1  {-----}
2  SOURCE:      SCOPE_3A.PAS
3
4  PURPOSE:     Demonstrates a simple MODULE and PROGRAM in VAXELN
5                Pascal with outer level declaration of a CONST and a
6                PROCEDURE declared in the program.
7
8  COMPILE:     $ EPASCAL /LIST /DEBUG SCOPE_3A
9
10 LINK:        $ LINK /DEBUG SCOPE_3A, ELN$:RTLSHARE /LIBRARY, -
11              _$ RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT SCOPE_3A
14
15 NOTES:       1) Command procedure SCOPE_3A.COM compiles, links and
16                builds this module into a VAXELN system or for
17                running under VMS
18 {-----}
19 MODULE SCOPE_3A [IDENT ('V1.000')];
20
21     CONST      { This outer level declaration is EXPORTed explicitly }
22     Greeting = 'A simple program to test VAXELN';
23 {-----}
24
25 PROGRAM SCOPE_3A (OUTPUT);
26
27     CONST
28     Greeting = 'This is an inner declaration of the same name';
29
30 {-----}
31 PROCEDURE Show_scope;
32
33     BEGIN
34     WRITELN ( 'This is from procedure SHOW_SCOPE about ',
35             'to write GREETING' );
36     WRITELN ( Greeting );
37     END { PROCEDURE Show_scope };
38
39 {-----}
40     BEGIN
41
42     WRITELN ( Greeting );
43     { prints "This is an inner declaration of the same name" }
44     Show_scope;
45
46 END { of PROGRAM SCOPE_3A }.
47 END { of MODULE SCOPE_3A };

```

EXAMPLES OF LANGUAGE EXTENSIONS

B.12.1 Running SCOPE_3A

SCOPE_3A can be built using the command procedure of the same name:

```
1 $      ! SCOPE_3A.COM
2 $      ! Command procedure to build the VAXELN module SCOPE_3A
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION  Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $      IF .NOT. Running_under_VMS THEN GOTO Compile
11 $ !
12 $      SET VERIFY
13 $      EPASCAL /LIST SCOPE_3A
14 $      LINK SCOPE_3A
15 $      GOTO Switch_off_verify
16 $ !
17 $ Compile:
18 $          @ELN_COMPILE_1  SCOPE_3A
19 $ Link:
20 $          @ELN_LINK_1     SCOPE_3A
21 $ System_build:
22 $          @ELN_EBUILD_1   SCOPE_3A
23 $ Switch_off_verify:
24 $      SET NOVERIFY
25 $      EXIT
```

The .DAT file used by EBUILD is listed below:

```
1      characteristic /noconsole
2      program SCOPE_3A /debug
```

When SCOPE_3A is run the output looks like this:

```
$ RUN SCOPE_3A
This is an inner declaration of the same name
This is from procedure SHOW_SCOPE about to write GREETING
This is an inner declaration of the same name
```

B.13 EXAMPLE OF SEPARATE - SEPARATE_PROG_1.PAS

Separating procedure bodies from headings enables compilation of developing sources to proceed without the availability of full procedures.

```

1  {-----}
2  SOURCE:      SEPARATE_PROG_1.PAS
3
4  PURPOSE:     Demonstrates a simple MODULE and PROGRAM in VAXELN
5                Pascal with outer level declaration of a PROCEDURE
6                The procedure body is in a source file compiled
7                separately
8
9  NEEDS:       1) SEPARATE_MOD_1.OBJ
10
11 COMPILE:      $ EPASCAL /LIST /DEBUG SEPARATE_MOD_1
12                $ EPASCAL /LIST /DEBUG SEPARATE_BODY_1, -
13                _$ SEPARATE_MOD_1/MOD
14                $ EPASCAL /LIST /DEBUG SEPARATE_PROG_1, -
15                _$ SEPARATE_MOD_1/MODULE
16
17 LINK:        $ LINK /DEBUG SEPARATE_PROG_1, SEPARATE_BODY_1, -
18                _$ ELN$:RTLSHARE /LIBRARY, -
19                _$ RTL /LIBRARY
20
21 BUILD:       $ EBUILD /NOEDIT SEPARATE_PROG_1
22
23 NOTES:       1) Command procedure SEPARATE_PROG_1.COM compiles,
24                links and builds this module into a system.
25                It may also be built to run under VMS using a
26                simple LINK command
27  {-----}
28 MODULE SEPARATE_PROG_1 [IDENT ('V1.000')];
29
30 INCLUDE
31     SEPARATE_MOD_1; { PROCEDURE Show_scope; SEPARATE; }
32
33 PROGRAM SEPARATE_PROG_1 (OUTPUT);
34
35     CONST
36         Greeting = 'This is an inner declaration of the same name';
37
38     BEGIN
39
40         WRITELN ( Greeting );
41         { prints "This is an inner declaration of the same name" }
42         Show_scope;
43
44     END { of PROGRAM SEPARATE_PROG_1 }.
45     END { of MODULE SEPARATE_PROG_1 };

```


EXAMPLES OF LANGUAGE EXTENSIONS

```
1  MODULE SEPARATE_MOD_1 [IDENT ('V1.000')];
2
3  PROCEDURE Show_scope; SEPARATE;
4
5  END; { MODULE SEPARATE_MOD_1 }
```

```
1  MODULE SEPARATE_BODY_1 [IDENT ('V1.000')];
2
3  {SOURCE:  SEPARATE_BODY_1.PAS
4   COMPILE: $ EPASCAL SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
5   }
6  PROCEDURE_BODY Show_scope;
7
8  CONST
9      Greeting = 'A simple program to test VAXELN';
10
11  BEGIN
12      WRITELN ( 'This is from procedure SHOW_SCOPE about ',
13              'to write GREETING' );
14      WRITELN ( Greeting );
15  END { PROCEDURE Show_scope };
16
17  END; { MODULE SEPARATE_BODY_1 }
```

B.13.1 Running SEPARATE_PROG_1

SEPARATE_PROG_1 can be built using the command procedure of the same name:

```

1  $      ! SEPARATE_PROG_1.COM
2  $      ! Command procedure to build the VAXELN
3  $      ! module SEPARATE_PROG_1
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $ !
8  $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile_defs
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST SEPARATE_MOD_1
15 $      EPASCAL /LIST SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
16 $      EPASCAL /LIST SEPARATE_PROG_1, SEPARATE_MOD_1 /MODULE
17 $      LINK SEPARATE_PROG_1, SEPARATE_BODY_1
18 $      GOTO Switch_off_verify
19 $ !
20 $ Compile_defs:
21 $      EPASCAL -
22         /LIST -
23         /DEBUG SEPARATE_MOD_1
24 $      EPASCAL -
25         /LIST -
26         /DEBUG SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
27 $ Compile:
28 $      EPASCAL -
29         /LIST -
30         /DEBUG SEPARATE_PROG_1, SEPARATE_MOD_1 /MODULE
31 $ Link:
32 $      LINK -
33         /DEBUG SEPARATE_PROG_1, SEPARATE_BODY_1 -
34         ,ELN$:RTLSHARE /LIBRARY -
35         ,ELN$:RTL /LIBRARY
36 $ System_build:
37 $      EBUILD -
38         /NOEDIT SEPARATE_PROG_1
39 $ Switch_off_verify:
40 $      SET NOVERIFY
41 $      EXIT

```

EXAMPLES OF LANGUAGE EXTENSIONS

The .DAT file used by EBUILD is listed below:

```
1 characteristic /noconsole
2 program SEPARATE_PROG_1 /debug
```

When SEPARATE_PROG_1 is run the output looks like this:

```
$ RUN SEPARATE_PROG_1
This is an inner declaration of the same name
This is from procedure SHOW_SCOPE about to write GREETING
A simple program to test VAXELN
```

B.14 EXAMPLE OF AGGREGATE INITIALIZATION - AGGREG_1.PAS

```

1  {-----}
2  SOURCE:      AGGREG_1.PAS
3
4  PURPOSE:     Demonstrates aggregate initialization on an array
5
6  COMPILE:     $ EPASCAL /LIST /DEBUG AGGREG_1
7
8  LINK:        $ LINK /DEBUG AGGREG_1 -
9               _$ ,ELN$:RTLSHARE /LIBRARY -
10              _$ ,RTL /LIBRARY
11
12 BUILD:       $ EBUILD /NOEDIT AGGREG_1
13
14 NOTES:       1) Command procedure AGGREG_1.COM compiles, links and
15               builds this module into a VAXELN system or for
16               running under VMS
17  {-----}
18  MODULE Aggregate [IDENT ('V1.000')];
19
20  TYPE
21      Chess_pieces = (QR,QN,QB,Q,K,KB,KN,KR,P,E);
22
23  { here is the aggregate initializer for the array - the technique
24    can be used on RECORDs as well.
25
26    The author has 'lifted' this one from the VAX Pascal Language
27    Reference Manual to whose authors he is grateful
28  }
29
30  VAR
31      Board      : ARRAY[1..8,QR..KR] OF Chess_pieces :=
32                  ((QR,QN,QB,Q,K,KB,KN,KR),
33                   (8 OF P),
34                   4 OF (8 OF E),
35                   (8 OF P),
36                   (QR,QN,QB,Q,K,KB,KN,KR));
37

```

EXAMPLES OF LANGUAGE EXTENSIONS

```

38 PROGRAM Chessboard (OUTPUT);
39
40 VAR
41     Rows    : 1..8;
42     Pieces  : QR..KR;
43
44 BEGIN
45     FOR Rows := 1 TO 8 DO
46         BEGIN
47             WRITELN;
48             WRITE(' ':25);
49             FOR Pieces := QR TO KR DO WRITE ( Board[Rows,Pieces] );
50         END;
51     WRITELN;
52
53 END {of PROGRAM}.
54 END {of MODULE};

```

B.14.1 Running AGGREG_1

This module can be built using AGGREG_1.COM:

```

1 $      ! AGGREG_1.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module AGGREG_1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST AGGREG_1
15 $      LINK AGGREG_1
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile:
19 $          @ELN_COMPILE_1 AGGREG_1
20 $ Link:
21 $          @ELN_LINK_1 AGGREG_1
22 $ System_build:
23 $          @ELN_EBUILD_1 AGGREG_1
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT

```

EXAMPLES OF LANGUAGE EXTENSIONS

The .DAT file used by EDEBUD is listed below:

```
1      characteristic /noconsole
2      program AGGREG_1 /debug
```

When the program is run the output looks like this:

```
$ RUN AGGREG_1
```

```
QR QN QB  Q  K KB KN KR
P  P  P  P  P  P  P  P
E  E  E  E  E  E  E  E
E  E  E  E  E  E  E  E
E  E  E  E  E  E  E  E
E  E  E  E  E  E  E  E
P  P  P  P  P  P  P  P
QR QN QB  Q  K KB KN KR
```


APPENDIX C

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

There are many examples supplied with this course that demonstrate modules, programs and processes but this appendix shows some simple examples.

C.1 EXAMPLE OF MODULE, PROGRAM AND PROCESS - MOD-PROG-PROC-1.PAS

This simple module shows the arrangement of a module, program and process. Note that to build this module you need the object library VAXELN-MODULES that is provided for this course. This library contains the definition of the procedure Check_status_and_report which is listed below for convenience. See programs ERRORS_1.PAS and ERRORS_2.PAS at the end of this appendix to show how this routine and VAXELN-supplied routines work

```
1  MODULE Check_status_and_report [IDENT ('V1.100')];
2
3  {
4      This module checks status returns and generates VMS-like error
5      messages when non-ODD status codes are returned
6
7      Modification history:
8
9      1) 09 February 1987: $OTSMSG line added to INCLUDE heading
10
11 }
12
```


EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

```
13  INCLUDE
14      $GET_MESSAGE_TEXT,
15      $ELNMSG,
16      $PASCALMSG,
17      $KERNELMSG,
18      $OTSMSG;          { Added 9-Feb-1987 }
19  VAR
20      Text_of_error      : VARYING_STRING (255);
21
22  PROCEDURE Check_status_and_report ( Returned_status : INTEGER;
23                                     Call_name : VARYING_STRING (80) );
24      BEGIN
25          IF NOT ODD ( Returned_status ) THEN
26              BEGIN
27                  WRITELN ( 'Unsuccessful call to: ',
28                           Call_name,
29                           ' status was: ',
30                           Returned_status : 1 );
31
32                  ELN$GET_STATUS_TEXT ( Returned_status,
33                                         [ STATUS$FACILITY,
34                                           STATUS$SEVERITY,
35                                           STATUS$IDENT,
36                                           STATUS$TEXT ],
37                                         Text_of_error );
38                  WRITELN ( Text_of_error );
39              END;
40      END;
41  END;
```

Now the listing of the module:

```

1  {
2  SOURCE:          MOD-PROG-PROC-1.PAS
3
4  PURPOSE:         Demonstrates a simple MODULE, PROGRAM and PROCESS
5                   in VAXELN
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG MOD-PROG-PROC-1, -
8                   _$ VAXELN-MODULES /LIBRARY
9
10 LINK:            $ LINK /DEBUG MOD-PROG-PROC-1, -
11                  _$ VAXELN-MODULES /LIBRARY, -
12                  _$ ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY /INCLUDE= -
13                  _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                  _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
15
16 BUILD:           $ EBUILD /NOEDIT MOD-PROG-PROC-1
17
18 NOTES:           1) Command procedure MOD-PROG-PROC-1.COM compiles,
19                  links and builds this module into a system
20
21 }
22
23 MODULE Create_process_01 [IDENT ('V1.000')];
24
25 INCLUDE
26     Check_status_and_report;
27
28 { These outer-level declarations of CONST and PROCESS_BLOCK are
29   EXPORTed implicitly
30 }
31     CONST
32         Master_process_greeting = 'This is from the master process';
33         Process_greeting        = 'This is from the created process';
34
35 {----- PROCESS BLOCK -----}
36
37 PROCESS_BLOCK Simple_process ( Value_passed : INTEGER );
38
39     BEGIN
40         WRITELN ( Process_greeting );
41         WRITELN ( 'Value passed to process is: ', Value_passed :1 );
42
43     END { PROCESS_BLOCK };
44

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

```

45 {----- PROGRAM BLOCK -----}
46
47     PROGRAM Process_create (OUTPUT);
48
49         VAR
50             Process_identity    : PROCESS;
51
52             Returned_status,
53             Number               : INTEGER;
54
55 {-----}
56
57     PROCEDURE Issue_master_process_greeting;
58
59     BEGIN
60         WRITELN ( Master_process_greeting );
61     END;
62
63 {-----}
64
65     PROCEDURE Create_process_and_wait;
66     BEGIN
67         CREATE_PROCESS ( Process_identity,
68                         Simple_process,
69                         Number,
70                         STATUS := Returned_status );
71
72         Check_status_and_report ( Returned_status,
73                                   'CREATE_PROCESS' );
74
75         WAIT_ANY ( Process_identity,
76                   STATUS := Returned_status );
77
78         Check_status_and_report ( Returned_status,
79                                   'WAIT_ANY' );
80
81     END { PROCEDURE DECLARATIONS };
82
83     { MAIN program start }
84
85     BEGIN
86         Issue_master_process_greeting;
87         Number := 1986;
88
89         Create_process_and_wait;
90
91     END {PROGRAM BLOCK}.
92     END {MODULE};

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

C.1.1.1 Running MOD-PROG-PROC-1.PAS

MOD-PROG-PROC-1.PAS can be built using the command procedure of the same name:

```
1 $      ! MOD-PROG-PROC-1.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module MOD-PROG-PROC-1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile_01:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG MOD-PROG-PROC-1, VAXELN-MODULES /LIBRARY
13 $ Link_01:
14 $      LINK -
15          /DEBUG MOD-PROG-PROC-1 -
16          ,VAXELN-MODULES /LIBRARY -
17          ,ELN$:RTLSHARE /LIBRARY -
18          ,ELN$:RTL /LIBRARY /INCLUDE= -
19          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
20          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
21 $ System_build:
22 $      EBUILD -
23          /NOEDIT MOD-PROG-PROC-1
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT
```

The .DAT file used by EBUILD is listed below:

```
1 characteristic /noconsole
2 program MOD-PROG-PROC-1 /debug
```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

C.2 EXAMPLE OF MODULE, PROGRAM AND PROCESS - MOD-PROG-PROC-1A.PAS

This is a version of the previous module with the declarations in a separate definitions module MOD-PROG-PROC-1A-DEFS.PAS

```
1 {  
2 SOURCE:      MOD-PROG-PROC-1A.PAS  
3  
4 PURPOSE:     Demonstrates a simple MODULE, PROGRAM and PROCESS  
5               in VAXELN  
6  
7 NEEDS:       VAXELN-MODULES.OLB  
8  
9 COMPILE:     $ EPASCAL /LIST /DEBUG MOD-PROG-PROC-1A, -  
10             _$ VAXELN-MODULES /LIB  
11  
12 LINK:       $ LINK /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIB, -  
13             _$ ELN$:RTL$SHARE /LIBRARY, RTL /LIBRARY /INCLUDE= -  
14             _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -  
15             _$ (OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)  
16  
17 BUILD:      $ EBUILD /NOEDIT MOD-PROG-PROC-1A  
18  
19 NOTES:      1) Command procedure MOD-PROG-PROC-1A.COM compiles,  
20               links and builds this module into a system  
21 }  
22 MODULE Create_process_01 [IDENT ('v1.000')];  
23  
24 INCLUDE MOD_PROG_PROC_1A_DEFS; { In library VAXELN-MODULES }  
25  
26 PROGRAM Process_create (OUTPUT);  
27  
28     { MAIN program start }  
29  
30     BEGIN  
31         Issue_master_process_greeting;  
32         Number := 1986;  
33         Create_process_and_wait;  
34  
35     END {PROGRAM}.  
36 END {MODULE};
```

The definition module looks like this:

```

1  MODULE MOD_PROG_PROC_1A_DEFS [IDENT ('V1.000')];
2  {
3      SOURCE:      MOD-PROG-PROC-1A-DEFS.PAS
4      COMPILE:     $ EPASCAL MOD-PROG-PROC-1A-DEFS.PAS, -
5                  _$ VAXELN-MODULES /LIBRARY
6
7      These outer-level declarations of CONST and PROCESS_BLOCK are
8      EXPORTed implicitly
9  }
10
11  INCLUDE
12      Check_status_and_report;
13
14  CONST
15      Master_process_greeting = 'This is from the master process';
16      Process_greeting        = 'This is from the created process';
17
18  VAR
19      Process_identity        : PROCESS;
20      Returned_status,
21      Number                  : INTEGER;
22
23  {----- PROCESS BLOCK -----}
24
25  PROCESS_BLOCK Simple_process ( Value_passed : INTEGER );
26
27      BEGIN
28          WRITELN ( Process_greeting );
29          WRITELN ( 'Value passed to process is: ', Value_passed :1 );
30
31  END {PROCESS_BLOCK};
32  {-----}
33
34  PROCEDURE Issue_master_process_greeting;
35
36      BEGIN
37          WRITELN ( Master_process_greeting );
38      END;
39

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

```
40 {-----}
41
42 PROCEDURE Create_process_and_wait;
43 BEGIN
44     CREATE_PROCESS ( Process_identity,
45                     Simple_process,
46                     Number,
47                     STATUS := Returned_status );
48
49     Check_status_and_report ( Returned_status,
50                             'CREATE_PROCESS' );
51
52     WAIT_ANY ( Process_identity,
53              STATUS := Returned_status );
54
55     Check_status_and_report ( Returned_status,
56                             'WAIT_ANY' );
57
58 END {PROCEDURE DECLARATIONS};
59 END {MODULE}.
```

C.2.1 Running MOD-PROG-PROC-1A

This module can be built using the command procedure of the same name:

```

1  $      ! MOD-PROG-PROC-1A.COM
2  $      ! Command procedure to build the VAXELN
3  $      ! module MOD-PROG-PROC-1A
4  $ !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $ !
8  $      SET VERIFY
9  $ !
10 $ Compile:
11 $      EPASCAL -
12          /LIST -
13          /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIB
14 $ Link:
15 $      LINK -
16          /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIBRARY -
17          ,ELN$:RTLSHARE /LIBRARY -
18          ,ELN$:RTL /LIBRARY /INCLUDE= -
19          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
20          OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
21 $ System_build:
22 $      EBUILD -
23          /NOEDIT MOD-PROG-PROC-1A
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT

```

The .DAT file looks like this:

```

1      characteristic /noconsole
2      program MOD-PROG-PROC-1A /debug

```


EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

C.3 EXAMPLE OF ERROR REPORTING - ERRORS_1.PAS

The program listed below shows how the ELN\$GET_STATUS_TEXT routine in the user-defined module Check_status_and_report generates VMS-like error messages:

```
1  {-----}
2  SOURCE:      ERRORS_1.PAS
3
4  PURPOSE:     To demonstrate error message handling using a user-
5                defined routine to print messages
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG ERRORS_1, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG ERRORS_1, VAXELN-MODULES /LIBRARY,
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT ERRORS_1
17
18 NOTES:       1) Command procedure ERRORS_1.COM compiles, links and
19                builds this module into a VAXELN system
20 -----}
21 MODULE Errors_01 [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report; { user-defined message writer }
25
26 {----- PROGRAM BLOCK -----}
27
28 PROGRAM Errors (OUTPUT);
29
30     VAR
31         Flag                : EVENT;
32         Returned_status      : INTEGER;
33
34     { MAIN program start }
35
36     BEGIN
37         CREATE_EVENT      (      Flag,
38                                EVENT$CLEARED,
39                                STATUS := Returned_status );
40
41         Check_status_and_report ( Returned_status,
42                                    'CREATE_EVENT' );
43
```

```

44         DELETE          (          Flag,
45                           STATUS := Returned_status );
46
47         Check_status_and_report ( Returned_status,
48                                   'DELETE' );
49
50 { Force error condition by waiting for non-existent object }
51
52         WAIT_ANY          (          Flag,
53                           STATUS := Returned_status );
54
55         Check_status_and_report ( Returned_status,
56                                   'WAIT_ANY' );
57
58 END { of PROGRAM }.
59 END { of MODULE };

```

C.3.1 Running ERRORS_1

This module can be built using the command procedure of the same name:

```

1  $      ! ERRORS_1.COM
2  $      ! Command procedure to compile and link the VAXELN
3  $      ! module ERRORS_1
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $ !
8  $      SET VERIFY
9  $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG ERRORS_1 -
13          ,VAXELN-MODULES /LIBRARY
14 $ Link:
15 $      LINK -
16          /DEBUG ERRORS_1 -
17          ,VAXELN-MODULES /LIBRARY -
18          ,ELN$:RTLSHARE /LIBRARY -
19          ,ELN$:RTL /LIBRARY /INCLUDE= -
20          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
21          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
22 $ System_build:
23 $      EBUILD -
24          /NOEDIT ERRORS_1
25 $ Switch_off_verify:
26 $      SET NOVERIFY
27 $      EXIT

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

The .DAT file looks like this:

```
1      characteristic /noconsole
2      program ERRORS_1 /debug
```

Output from ERRORS_1:

```
SH SYS
! Available: Pages: 1436, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:19.23 Total:  0 00:00:19.36
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program ERRORS_1, priority 16 is waiting.
!
GO
! Job 5, process 1, program ERRORS_1 running.
!Unsuccessful call to: WAIT_ANY status was: 31804
!%KERNEL-F-BAD_VALUE, bad parameter value
! Job 5, process 1, program ERRORS_1 has exited.
!
EXIT
```

C.4 EXAMPLE OF ERROR REPORTING - ERRORS_2.PAS

The program listed below shows how the VMS MESSAGE utility may be used

```

1  {-----
2  SOURCE:          ERRORS_2.PAS
3
4  PURPOSE:         To demonstrate error message handling
5
6  NEEDS:           ERRORS_2_MSG.MSG
7
8  COMPILE:         $ EPASCAL /LIST /DEBUG ERRORS_2,
9                  _$ ELN$:RTLOBJECT /LIBRARY
10
11                  $ MESSAGE /OBJECT ERRORS_2_MSG
12
13 LINK:            $ LINK /DEBUG ERRORS_2, ERRORS_2_MSG, -
14                  _$ ELN$:RTLSHARE /LIBRARY, -
15                  _$ ELN$:RTL /LIBRARY /INCLUDE= -
16                  _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
17                  _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
18
19 BUILD:           $ EBUILD /NOEDIT ERRORS_2
20
21 NOTES:           1) Command procedure ERRORS_2.COM compiles, links and
22                  builds this module into a VAXELN system
23 -----}
24 MODULE Errors_02 [IDENT ('V1.000')];
25
26 INCLUDE
27     $GET_MESSAGE_TEXT;
28
29 VAR
30     { the symbol "User_obj_deleted" is defined in ERRORS_2_MSG.MSG }
31
32     KER$_BAD_VALUE,
33     User_Obj_deleted : [EXTERNAL, VALUE] INTEGER;
34
35 { this SET type is defined in $GET_MESSAGE_TEXT }
36
37     Message_fields  : SET OF Get_status_fields := [];
38
39

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

```

40 {----- PROGRAM BLOCK -----}
41
42 PROGRAM Errors_02 (OUTPUT);
43
44     VAR
45         Flag                : EVENT;
46         Returned_status     : INTEGER;
47         Mess_text           : VARYING_STRING(255);
48
49 { MAIN program start }
50
51     BEGIN
52         CREATE_EVENT      (      Flag,
53                               EVENT$CLEARED,
54                               STATUS := Returned_status );
55
56         IF NOT ODD ( Returned_status )
57             THEN WRITELN ( 'CREATE_EVENT problem, status was: ',
58                             Returned_status:1 );
59
60         DELETE            (      Flag,
61                               STATUS := Returned_status );
62
63         IF NOT ODD ( Returned_status )
64             THEN WRITELN ( 'DELETE problem, status was: ',
65                             Returned_status:1 );
66
67 { Force error condition by waiting for non-existent object }
68
69         WAIT_ANY          (      Flag,
70                               STATUS := Returned_status );
71
72         IF Returned_status = KER$_BAD_VALUE
73             THEN
74                 BEGIN
75                     ELN$GET_STATUS_TEXT ( User_obj_deleted,
76                                             Message_fields,
77                                             Mess_text );
78                     WRITELN ( Mess_text );
79                 END;
80
81     END { of PROGRAM }.
82 END { of MODULE };

```

C.4.1 Running ERRORS_2

This module can be built using the command procedure of the same name:

```

1 $      ! ERRORS_2.COM
2 $      ! Command procedure to compile and link the VAXELN
3 $      ! module ERRORS_2
4 $      !
5 $      !
6 $ !
7 $      ON ERROR THEN GOTO Switch_off_verify
8 $      SET DEFAULT Default_directory
9 $ !
10 $      SET VERIFY
11 $ Compile:
12 $      EPASCAL -
13          /LIST -
14          /DEBUG ERRORS_2 -
15          , ELN$:RTLOBJECT /LIBRARY
16 $ Message:
17     MESSAGE -
18         /OBJECT ERRORS_2_MSG
19 $ Link:
20 $      LINK -
21          /DEBUG ERRORS_2, ERRORS_2_MSG -
22          ,ELN$:RTLSHARE /LIBRARY -
23          ,ELN$:RTL /LIBRARY /INCLUDE= -
24          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
25          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
26 $ System_build:
27 $      EBUILD -
28          /NOEDIT ERRORS_2
29 $ Switch_off_verify:
30 $      SET NOVERIFY
31 $      EXIT

```

EXAMPLES OF MODULES, PROGRAMS AND PROCESSES

The file ERRORS_2_MSG.MSG looks like this:

```
1          .TITLE  VAXELN_demo
2
3          .FACILITY          USER,  1
4
5
6          .SEVERITY          SEVERE
7          OBJ_DELETED < Wait failed because the object did not exist >
8
9          .END
```

The .DAT file looks like this:

```
1      characteristic /noconsole /nonetwork /nofile /noserver
2      program ERRORS_2 /debug
```

APPENDIX D

EXAMPLES OF SYNCHRONIZATION AND TIME

This appendix deals with examples using the VAXELN time and synchronization routines.

D.1 EXAMPLE OF VAXELN TIME ROUTINES - TIME_1.PAS

This first module uses all the VAXELN time routines. The system clock is set and then the time is read and manipulated. By default the system time starts at 17-NOV-1858 00:00:00.00. Don't forget the EDEBUG command SET TIME allows setting of the VAXELN system time.

```
1 {
2 SOURCE:      TIME_1.PAS
3
4 PURPOSE:     Demonstrates the time routines in VAXELN
5
6 COMPILE:     $ EPASCAL /LIST /DEBUG TIME_1
7
8 LINK:        $ LINK /DEBUG TIME_1, -
9              _$ ELN$:RTLSHARE /LIBRARY, -
10             _$ RTL /LIBRARY
11
12 BUILD:      $ EBUILD /NOEDIT TIME_1
13
14 NOTES:      1) Command procedure TIME_1.COM compiles, links and
15              builds this module into a system
16
17 }
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

18 MODULE All_times [IDENT ('V1.000')];
19
20 CONST
21     Interval                      = '0007 12:30:30.00';
22 VAR
23     From_users_input_time         : VARYING_STRING (23);
24
25     At_this_moment,
26     Binary_interval,
27     New_time,
28     The_current_binary_time_is    : LARGE_INTEGER;
29
30     Status_from_call_to_GET_TIME,
31     Status_from_call_to_GET_TIME_1,
32     Status_from_call_to_SET_TIME  : INTEGER;
33
34     Time_elements                 : TIME_RECORD;
35
36 { ----- }
37
38 PROCEDURE Prompt_user_for_time;
39
40 BEGIN
41     WRITELN ( 'Please enter time e.g. ',
42              '12-Jan-1987 15:47:30.00 > ' );
43     READLN ( From_users_input_time );
44 END;
45
46 { ----- }
47
48 PROCEDURE And_get_the_binary_time;
49
50 BEGIN
51     The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
52 END;
53
54 { ----- }
55
56 PROCEDURE Set_the_system_time;
57
58 BEGIN
59     SET_TIME      ( The_current_binary_time_is,
60                    STATUS := Status_from_call_to_SET_TIME );
61
62     IF
63         NOT ODD ( Status_from_call_to_SET_TIME )
64     THEN
65         WRITELN ( 'Call to SET_TIME failed with status: ',
66                  Status_from_call_to_SET_TIME
67                  );
68 END;
69

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

70 { ----- }
71
72 PROCEDURE And_read_time_fields;
73
74 BEGIN
75     GET_TIME      ( At_this_moment,
76                   STATUS := Status_from_call_to_GET_TIME );
77
78     IF
79         NOT ODD ( Status_from_call_to_GET_TIME )
80     THEN
81         WRITELN ( 'Call to GET_TIME failed with status: ',
82                 Status_from_call_to_GET_TIME
83                 );
84
85         Time_elements := TIME_FIELDS      ( At_this_moment );
86
87         WRITELN ( 'The current TIME_FIELDS are: ' );
88
89         WITH Time_elements DO
90             WRITELN ( Hour      :2, ': ',
91                     Minute    :2, ': ',
92                     Second     :2, '. ',
93                     Hundredth  :2, ' on ',
94                     Day        :2, '/',
95                     Month      :2, '/',
96                     Year       :4
97                     );
98     END;
99
100 { ----- }
101
102 PROCEDURE Get_the_binary_for_interval;
103
104 BEGIN
105     Binary_interval := TIME_VALUE ( Interval );
106 END;
107
108 { ----- }
109
110 PROCEDURE Add_interval_to_current_time;
111
112 BEGIN
113     { NOTE: delta times are held as negative integers. To
114       advance by an interval SUBTRACT that interval! }
115

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

116     GET_TIME      ( At_this_moment,
117                     STATUS := Status_from_call_to_GET_TIME_1 );
118
119     IF
120         NOT ODD ( Status_from_call_to_GET_TIME_1 )
121     THEN
122         WRITELN ( 'Call to GET_TIME failed with status: ',
123                 Status_from_call_to_GET_TIME_1
124                 );
125
126     New_time := At_this_moment - Binary_interval;
127
128     END;
129
130 { ----- }
131
132     PROCEDURE And_find_new_absolute_time;
133
134     BEGIN
135         WRITELN ( 'The new time formed by adding an interval of: ',
136                 Interval, ' is: ' );
137         WRITELN ( TIME_STRING ( New_time ) );
138     END;
139
140 { ----- }
141
142     PROGRAM Show_times ( INPUT, OUTPUT );
143
144     BEGIN
145         Prompt_user_for_time;
146         And_get_the_binary_time;
147
148         Set_the_system_time;
149         And_read_time_fields;
150
151         Get_the_binary_for_interval;
152         Add_interval_to_current_time;
153
154         And_find_new_absolute_time;
155
156     END { of PROGRAM };
157     END { of MODULE };

```

D.1.1.1 Running TIME_1

Use the command procedure TIME_1.COM to build this program:

```

1 $      ! TIME_1.COM
2 $      ! Command procedure to build the VAXELN module TIME_1
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      SET VERIFY
8 $ Compile_Time:
9 $      EPASCAL -
10             /LIST -
11             /DEBUG TIME_1
12 $ Link_Time:
13 $      LINK -
14             /DEBUG TIME_1 -
15             ,ELN$:RTLSHARE /LIBRARY -
16             ,ELN$:RTL /LIBRARY
17 $ System_build:
18 $      EBUILD -
19             /NOEDIT TIME_1
20 $ Switch_off_verify:
21 $      SET NOVERIFY
22 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole
2      program TIME_1 /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from TIME_1:

SHOW SYSTEM

```
! Available: Pages: 17869, Page table slots: 51, Pool blocks: 273
! Time since SET TIME: Idle:  0 00:00:50.05 Total:  0 00:00:50.21
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program TIME_1, priority 16 is waiting.
!
```

GO

```
! Job 5, process 1, program TIME_1 running.
! Please enter time e.g. 12-Jan-1987 15:47:30.00 > 02-Mar-1987 14:36:35.00
! The current TIME_FIELDS are:
! 14:36:35. 0 on  2/ 3/1987
! The new time formed by adding an interval of: 0007 12:30:30.00 is:
! 10-MAR-1987 03:07:05.15
! Job 5, process 1, program TIME_1 has exited.
!
```

exit

D.2 EXAMPLE OF VAXELN RUNNING FORTRAN - TIME_2.PAS

This module runs a FORTRAN program to display the date and time having been set by a Pascal program first.

```

1  {
2  SOURCE:          TIME_2.PAS
3
4  PURPOSE:         Sets the system time then creates a job to run a
5                    program written in VAX FORTRAN that returns:
6                    a) day
7                    b) month
8                    c) year
9                    d) week
10                   e) year day
11                   f) century day
12                   for that date
13
14  COMPILE:         $ EPASCAL /LIST /DEBUG TIME_2
15
16                   $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY
17
18  LINK:            $ LINK /DEBUG TIME_2, -
19                   _$ ELN$:RTLSHARE /LIBRARY, -
20                   _$ RTL /LIBRARY
21
22                   $ LINK /DEBUG /NOSYSLIB TODAY, -
23                   _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
24                   _$ RTL /LIBRARY
25
26  BUILD:           $ EBUILD /NOEDIT TIME_2
27
28  NOTES:           1) Command procedure TIME_2.COM compiles, links and
29                   builds these programs into a system
30
31  }
32  MODULE Set_system_time [IDENT ('V1.000')];
33
34  VAR
35      From_users_input_time      : VARYING_STRING (23);
36      Status_from_call_to_SET_TIME : INTEGER;
37      The_current_binary_time_is  : LARGE_INTEGER;
38
39  { ----- }
40
41  PROCEDURE Prompt_user_for_time;
42
43      BEGIN
44          WRITELN ( 'Please enter time e.g. ',
45                  '12-Dec-1986 15:47:30.00 > ' );
46          READLN  ( From_users_input_time );
47      END;
48

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

49 { ----- }
50
51 PROCEDURE And_get_the_binary_time;
52
53 BEGIN
54     The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
55 END;
56
57 { ----- }
58
59 PROCEDURE Set_the_system_time;
60
61 BEGIN
62     SET_TIME      ( The_current_binary_time_is,
63                     STATUS := Status_from_call_to_SET_TIME );
64
65     IF NOT ODD    ( Status_from_call_to_SET_TIME ) THEN
66         WRITELN ( 'Call to SET_TIME failed with status: ',
67                   Status_from_call_to_SET_TIME );
68 END;
69
70 { ----- }
71
72 PROGRAM Set_system_time ( INPUT, OUTPUT );
73
74 VAR
75     Status_returned : INTEGER;
76     Jobs_port       : PORT;
77
78 BEGIN
79     Prompt_user_for_time;
80     And_get_the_binary_time;
81     Set_the_system_time;
82
83     CREATE_JOB ( Jobs_port,
84                  'TODAY',
85                  STATUS := Status_returned );
86     IF NOT ODD ( Status_returned ) THEN
87         WRITELN ( 'Call to CREATE_JOB failed with status: ',
88                   Status_returned );
89
90 END { of PROGRAM }.
91 END { of MODULE };

```

EXAMPLES OF SYNCHRONIZATION AND TIME

The source code for the FORTRAN program TODAY is listed below.

```

1  *
2  *   MODULE:          TODAY.FOR
3  *
4  *   PURPOSE:        To calculate:
5  *                   a) Day of week
6  *                   b) Day of year
7  *                   c) Day of century
8  *                   d) Week number (crude version)
9  *                   for any date after today.
10 *                   This version is fixed to pick up today's date
11 *                   but is capable of taking any date from 1770
12 *
13 *   COMPILE:         $ FORTRAN /TODAY
14 *   LINK:            $ LINK /TODAY
15 *
16 *   RESTRICTIONS:    Works until 31-Dec-2086
17 *
18
19   IMPLICIT NONE
20
21   INTEGER            Month_length(12) /31, 28, 31, 30, 31, 30,
22   1                  31, 31, 30, 31, 30, 31/
23
24   INTEGER            Verbal_day_value(7) /1, 2, 3, 21, 22, 23, 31/
25
26   CHARACTER *9       Month_names(12) /'January',    'February',
27   1                  'March',      'April',
28   1                  'May',        'June',
29   1                  'July',       'August',
30   1                  'September', 'October',
31   1                  'November',  'December'/
32
33   CHARACTER *9       Day_of_week(0:6) /'Sunday',    'Monday',
34   1                  'Tuesday',   'Wednesday',
35   1                  'Thursday',  'Friday',
36   1                  'Saturday'/
37
38   CHARACTER *2       Date_suffix(4)  /'st', 'nd', 'rd', 'th'/
39
40   CHARACTER *8       Time_now

```


EXAMPLES OF SYNCHRONIZATION AND TIME

```

41
42     INTEGER          Y,
43     1                K,
44     1                Day,
45     1                Day_suffix_no,
46     1                Day_of_year,
47     1                Week,
48     1                Month,
49     1                Year,
50     1                Century,
51     1                Start_of_century,
52     1                Century_day,
53     1                Grand_day_count,
54     1                Weekday
55
56     LOGICAL          Leap_year
57
58 *==>> Statement function to determine leap year/no leap year
59
60     Leap_year(Y) = MOD (Y, 4) .EQ. 0 .AND.
61     1            MOD (Y, 100) .NE. 0 .OR.
62     1            MOD (Y, 400) .EQ. 0
63
64 *==>> Get numeric values of date;
65 *==>> Program will work correctly until 31 December 2086
66
67     CALL IDATE ( Month, Day, Year )
68
69     IF (Year .LT. 87) THEN
70         Year = Year + 2000
71     ELSE
72         Year = Year + 1900
73     END IF
74
75 *==>> Initialize year day value;
76 *==>> Find total days so far this year
77
78     Day_of_year = Day
79
80     DO K = 1, Month-1, 1
81         Day_of_year = Day_of_year + Month_length(K)
82     END DO
83
84     IF (Month .GT. 2 .AND. Leap_year(Year)) Day_of_year = Day_of_year + 1
85
86 *==>> Find week for current year. NOTE: This is not the
87 *==>> internationally agreed definition of week
88
89     Week = (Day_of_year - 1) / 7 + 1
90

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

91 *==>> Get century number BUT check not e.g. 1900 - last year of
92 *==>> 19th century
93
94     Century_day = Day_of_year
95
96     Century = Year / 100
97     IF (MOD (Year, 100) .EQ. 0) Century = Century - 1
98     Start_of_century = Century * 100 + 1
99
100    DO K = Start_of_century, Year-1, 1
101    IF (Leap_year(K)) THEN
102        Century_day = Century_day + 366
103    ELSE
104        Century_day = Century_day + 365
105    END IF
106    END DO
107
108 *==>> Initialize total day count;
109 *==>> Total days from start date to last year
110 *==>> N.B 1 Jan 1770 was a MONDAY
111
112    Grand_day_count = Day_of_year
113
114    DO K = 1770, Year-1, 1
115    IF (Leap_year(K)) THEN
116        Grand_day_count = Grand_day_count + 366
117    ELSE
118        Grand_day_count = Grand_day_count + 365
119    END IF
120    END DO
121
122 *==>> Find day of week from total day count;
123 *==>> Determine which of "st", "nd", "rd" or "th" suffixes is
124 *==>> required
125
126    Weekday = MOD (Grand_day_count, 7)
127
128    Day_suffix_no = 4
129
130    DO K = 1, 7, 1
131    IF (Day .EQ. Verbal_day_value(K)) Day_suffix_no = MOD (Day, 10)
132    END DO
133
134    CALL Time ( Time_now )
135

```

EXAMPLES OF SYNCHRONIZATION AND TIME

D.2.1 Running TIME_2

Use the command procedure TIME_2.COM to build this system:

```

1 $      ! TIME_2.COM
2 $      ! Command procedure to build the EPASCAL module TIME_2
3 $      ! and the VAX FORTRAN program TODAY and build them into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile_Time_2:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG TIME_2
13 $ Compile_Today:
14 $      FORTRAN -
15          /LIST -
16          /NOOPTIMIZE -
17          /DEBUG TODAY
18 $ Link_Time_2:
19 $      LINK -
20          /DEBUG TIME_2 -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY
23 $ Link_Today:
24 $      LINK -
25          /DEBUG -
26          /NOSYSLIB TODAY -
27          ,ELN$:FRTLOBJECT /LIBRARY -
28          ,ELN$:RTLSHARE /LIBRARY -
29          ,ELN$:RTL /LIBRARY
30 $ System_build:
31 $      EBUILD -
32          /NOEDIT TIME_2
33 $ Switch_off_verify:
34 $      SET NOVERIFY
35 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole
2      program TIME_2 /debug
3      program TODAY /norun /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from TIME_2:

```
SHOW SYSTEM
! Available: Pages: 17828, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:13.15 Total:  0 00:00:13.32
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program TIME_2, priority 16 is waiting.
!
GO
! Job 5, process 1, program TIME_2 running.
! Please enter time e.g. 12-Dec-1986 15:47:30.00 > 02-Mar-1987 14:39:00.00
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]TODAY.EXE;1
!
! Job 6, process 1, program TODAY needs attention.
! Module TODAY$MAIN
! 18:
!>>19: IMPLICIT NONE
! 20:
! 21: INTEGER          Month_length(12) /31, 28, 31, 30, 31, 30,
! 22: 1                31, 31, 30, 31, 30, 31/
! 23:
! Job 5, process 1, program TIME_2 has exited.
!
GO
! Job 6, process 1, program TODAY running.
!
!      14:39:08 Monday      2nd March      1987  W  9, Y  61, C 31472
!
! Job 6, process 1, program TODAY has exited.
!
exit
```

D.3 EXAMPLE OF VAXELN RUNNING FORTRAN - TIME_2A.PAS

This module is a version of TIME_2. Instead of creating a job to run the FORTRAN it calls INITIALIZATION_DONE. The kernel then creates a job from which to run the FORTRAN program

```

1  {
2  SOURCE:          TIME_2A.PAS
3
4  PURPOSE:         This program is a variation on TIME_2. This version
5                   calls INITIALIZATION_DONE instead of creating a job
6                   to cause TODAY to run.
7
8                   The program TODAY written in VAX FORTRAN returns:
9                   a) day
10                  b) month
11                  c) year
12                  d) week
13                  e) year day
14                  f) century day
15                  for the date
16
17  COMPILE:         $ EPASCAL /LIST /DEBUG TIME_2A
18
19                  $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY
20
21  LINK:           $ LINK /DEBUG TIME_2A, -
22                  _$ ELN$:RTLSHARE /LIBRARY, -
23                  _$ RTL /LIBRARY
24
25                  $ LINK /DEBUG /NOSYSLIB TODAY, -
26                  _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
27                  _$ RTL /LIBRARY
28
29  BUILD:          $ EBUILD /NOEDIT TIME_2A
30
31  NOTES:          1) Command procedure TIME_2A.COM compiles, links and
32                  builds these programs into a system
33
34  }
35  MODULE Set_system_time [IDENT ('V1.000')];
36
37  VAR
38      From_users_input_time      : VARYING_STRING (23);
39      Status_from_call_to_SET_TIME : INTEGER;
40      The_current_binary_time_is  : LARGE_INTEGER;
41
42  { ----- }
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

43
44 PROCEDURE Prompt_user_for_time;
45
46 BEGIN
47     WRITELN ( 'Please enter time e.g. ',
48               '12-DEC-1986 15:47:30.00 > ' );
49     READLN ( From_users_input_time );
50 END;
51
52 { ----- }
53
54 PROCEDURE And_get_the_binary_time;
55
56 BEGIN
57     The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
58 END;
59
60 { ----- }
61
62 PROCEDURE Set_the_system_time;
63
64 BEGIN
65     SET_TIME      ( The_current_binary_time_is,
66                     STATUS := Status_from_call_to_SET_TIME );
67
68     IF NOT ODD ( Status_from_call_to_SET_TIME ) THEN
69         WRITELN ( 'Call to SET_TIME failed with status: ',
70                   Status_from_call_to_SET_TIME );
71 END;
72
73 { ----- }
74
75 PROGRAM Set_system_time ( INPUT, OUTPUT );
76
77 VAR
78     Status_returned : INTEGER;
79     Jobs_port       : PORT;
80
81 BEGIN
82     Prompt_user_for_time;
83     And_get_the_binary_time;
84     Set_the_system_time;
85
86     INITIALIZATION_DONE ( STATUS := Status_returned );
87     IF NOT ODD ( Status_returned ) THEN
88         WRITELN ( 'Call to INITIALIZATION_DONE ',
89                   ' failed with status: ', Status_returned );
90
91 END { of PROGRAM }.
92 END { of MODULE };

```

D.3.1 Running TIME_2A

Use the command procedure TIME_2A.COM to build this system:

```

1 $      ! TIME_2A.COM
2 $      ! Command procedure to build the EPASCAL module TIME_2A
3 $      ! and the VAX FORTRAN program TODAY and build them into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile_Time_2A:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG TIME_2A
13 $ Compile_Today:
14 $      FORTRAN -
15          /LIST -
16          /NOOPTIMIZE -
17          /DEBUG TODAY
18 $ Link_Time_2A:
19 $      LINK -
20          /DEBUG TIME_2A -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY
23 $ Link_Today:
24 $      LINK -
25          /DEBUG -
26          /NOSYSLIB TODAY -
27          ,ELN$:FRTLOBJECT /LIBRARY -
28          ,ELN$:RTLSHARE /LIBRARY -
29          ,ELN$:RTL /LIBRARY
30 $ System_build:
31 $      EBUILD -
32          /NOEDIT TIME_2A
33 $ Switch_off_verify:
34 $      SET NOVERIFY
35 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1 characteristic /noconsole
2 program TIME_2A /initialize /debug
3 program TODAY /debug

```

The difference here is the absence of the /norun qualification for TODAY

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from TIME_2A:

```
SH SYS
! Available: Pages: 17826, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:07.59 Total:  0 00:00:07.73
! Time used by past jobs: 0
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program TIME_2A, priority 16 is waiting.
!
GO
! Job 4, process 1, program TIME_2A running.
! Please enter time e.g. 12-DEC-1986 15:47:30.00 > 02-MAR-1987 14:53:05.00
! Job 4, process 1, program TIME_2A has exited.
!
! Job 6, process 1, program TODAY running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]TODAY.EXE;3
!
! Job 6, process 1, program TODAY needs attention.
! Module TODAY$MAIN
! 18:
!>>19: IMPLICIT NONE
! 20:
! 21: INTEGER          Month_length(12) /31, 28, 31, 30, 31, 30,
! 22: 1                31, 31, 30, 31, 30, 31/
! 23:
!
GO
! Job 6, process 1, program TODAY running.
!
! 14:53:09 Monday      2nd March      1987  W  9, Y  61, C 31472
!
! Job 6, process 1, program TODAY has exited.
!
EXIT
```

D.4 EXAMPLE OF VAXELN CALLING VMS FOR THE TIME - TIME_3.PAS

This program connects with VMS using a circuit. It collects the local VAX's node number from a program argument established at EBUILD time - command procedure details later. On connection with VMS a command procedure TIME.COM is executed in the default DECnet account of that VAX node. The time string obtained by the command procedure is returned to the VAXELN job on the circuit established. The VAXELN system time is then set from that time string.

After completion of the time setting the FORTRAN program TODAY is run from a newly-created job, as before.

```

1  {-----
2  SOURCE:          TIME_3.PAS
3
4  PURPOSE:         To demonstrate request for time from adjacent VMS node
5                    setting of that time on VAXELN and running of a
6                    FORTRAN program in a separate job
7
8                    APPLICATION11.PAS supplied with VAXELN shows the
9                    mechanics of a request of VMS for a time string
10
11  COMPILE:         $ EPASCAL /LIST /DEBUG TIME_3
12
13                   $ FORTRAN /NOOPTIMIZE /LIST /DEBUG TODAY
14
15  LINK:           $ LINK /DEBUG /NOSYSLIB TODAY, -
16                   _$ ELN$:FRTLOBJECT /LIBRARY, -
17                   _$ ELN$:RTLSHARE /LIBRARY, -
18                   _$ ELN$:RTL /LIBRARY
19
20  LINK:           $ LINK /DEBUG TIME_3, -
21                   _$ ELN$:RTLSHARE /LIBRARY, -
22                   _$ ELN$:RTL /LIBRARY
23
24  BUILD:          $ EBUILD /NOEDIT TIME_3
25
26  NEEDS:          TIME.COM in the default DECnet account.
27                  TIME.COM looks like this
28
29                  $ OPEN /WRITE Time_data SYS$NET
30                  $ Current_time = F$TIME()
31                  $ WRITE Time_data Current_time
32                  $ CLOSE Time_data
33                  $ EXIT
34
35  NOTES:          1) Command procedure TIME_3.COM compiles, links
36                  and builds this module into a VAXELN system
37  -----}
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

38  MODULE Get_VMS_time [IDENT ('V1.000')];
39
40  CONST
41      LF                = '(10);    { line-feed }
42      Argument_position = 3;
43
44  VAR
45      VAXELN_port,
46      New_jobs_port   : PORT;
47      VMS_message     : MESSAGE;
48      Destination     : VARYING_STRING(15);
49      VMS_time_string : ^STRING(32);
50      Actual_time     : STRING(32);
51      Current_time    : LARGE_INTEGER;
52      Node_number     : VARYING_STRING (7);
53      Status_returned : INTEGER;
54
55  {-----}
56
57  PROGRAM Get_VMS_time;
58
59  BEGIN
60
61  { find my job's port value instead of creating an additional
62    port }
63
64  JOB_PORT ( VAXELN_port,
65            STATUS := Status_returned );
66
67      IF NOT ODD ( Status_returned ) THEN
68          WRITELN ( 'JOB_PORT status was: ', Status_returned :1 );
69
70  { pick up the node number from the argument passed from the .DAT
71    file at EBUILD and concatenate it with the object name TIME }
72
73  Node_number := PROGRAM_ARGUMENT ( Argument_position );
74  Destination := Node_number + '::~TIME';
75
76  { create the link with the VMS node whose number has just been
77    collected }
78
79  WRITELN ( LF, ' 1) About to connect with VMS node: ',
80            Node_number, '...' );
81
82  CONNECT_CIRCUIT ( VAXELN_port,
83                  DESTINATION_NAME := Destination,
84                  STATUS := Status_returned );
85
86      IF NOT ODD ( Status_returned ) THEN
87          WRITELN ( 'CONNECT_CIRCUIT status was: ', Status_returned :1
88

```

```

89 { wait for VMS to respond with the time string }
90
91 WAIT_ANY ( VAXELN_port,
92           STATUS := Status_returned );
93
94     IF NOT ODD ( Status_returned ) THEN
95         WRITELN ( 'WAIT_ANY status was: ', Status_returned :1 );
96
97 RECEIVE ( VMS_message,
98           VMS_time_string,
99           VAXELN_port,
100          STATUS := Status_returned );
101
102     IF NOT ODD ( Status_returned ) THEN
103         WRITELN ( 'RECEIVE status was: ', Status_returned :1 );
104
105 DISCONNECT_CIRCUIT ( VAXELN_port,
106                     STATUS := Status_returned );
107
108     IF NOT ODD ( Status_returned ) THEN
109         WRITELN ( 'DISCONNECT_CIRCUIT status was: ',
110                 Status_returned :1 );
111
112 WRITELN ( LF, ' 2) Successful connection with VMS node' );
113
114 { parse the received string for a 23 byte absolute time }
115
116 Actual_time := SUBSTR ( VMS_time_string^, 1, 23 );
117 Current_time := TIME_VALUE ( Actual_time );
118
119 WRITELN ( LF, ' 3) About to set system time...' );
120
121 SET_TIME ( Current_time,
122           STATUS := Status_returned );
123
124     IF NOT ODD ( Status_returned ) THEN
125         WRITELN ( 'SET_TIME status was: ',
126                 Status_returned :1 );
127
128 WRITELN ( LF, ' 4) Creating job to run VAX FORTRAN program', LF );
129
130 CREATE_JOB ( New_jobs_port,
131             'TODAY',
132             STATUS := Status_returned );
133
134     IF NOT ODD ( Status_returned ) THEN
135         WRITELN ( 'CREATE_JOB status was: ', Status_returned :1 );
136
137 END {of PROGRAM}.
138 END {of MODULE };

```

EXAMPLES OF SYNCHRONIZATION AND TIME

The FORTRAN program is identical with that used in the earlier examples.

D.4.1 Running TIME_3

Use the command procedure TIME_3.COM to build this system. This command procedure uses an embedded command procedure that collects the node number of the host VAX where the system is being built. TIME_3_DAT.COM collects the node number using lexical function F\$GETSYI then builds the correct .DAT file. In this way the program does not need to be changed every time the node changes.

```
1 $      ! TIME_3.COM
2 $      ! Command procedure to build the EPASCAL module
3 $      ! TIME_3 and the VAX FORTRAN program TODAY and build them
4 $      ! into a system
5 $      !
6 $      ON ERROR THEN GOTO Switch_off_verify
7 $      SET DEFAULT Default_directory
8 $ !
9 $      SET VERIFY
10 $ Compile_Time_3:
11 $      EPASCAL -
12          /LIST -
13          /DEBUG TIME_3
14 $ Compile_Today:
15 $      FORTRAN -
16          /LIST -
17          /NOOPTIMIZE -
18          /DEBUG TODAY
19 $ Link_Time_3:
20 $      LINK -
21          /DEBUG TIME_3 -
22          ,ELN$:RTLSHARE /LIBRARY -
23          ,ELN$:RTL /LIBRARY
24 $ Link_Today:
25 $      LINK -
26          /DEBUG -
27          /NOSYSLIB TODAY -
28          ,ELN$:FRTLOBJECT /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY
31 $ !
32 $      SET NOVERIFY
33 $ Build_the_DAT_file:
34 $          @TIME_3_DAT
35 $      SET VERIFY
36 $ System_build:
37 $      EBUILD -
38          /NOEDIT TIME_3
39 $ Switch_off_verify:
40 $      SET NOVERIFY
41 $      EXIT
```

EXAMPLES OF SYNCHRONIZATION AND TIME

The command procedure TIME_3_DAT.COM looks like this:

```

1 $ !
2 $ !      MODULE:          TIME_3_DAT.COM
3 $ !      PURPOSE:        To build the .DAT file for TIME_3 automatically
4 $ !                        by grabbing the node area and node number using
5 $ !                        F$GETSYI the user can run the build without
6 $ !                        needing to know local node details
7 $ !
8 $ !      Finished file must look like this:
9 $ !
10 $ !      characteristic /noconsole
11 $ !      program TIME_3 /debug /argument=( "", "", ""1.241"" )
12 $ !      program TODAY /norun /debug
13 $ !
14 $      OPEN -
15          /WRITE -
16          Data_file -
17          TIME_3.DAT
18 $ !
19 $ Set_up_symbols:
20 $      Area      = F$GETSYI ( "Node_area" )
21 $      Node_num  = F$GETSYI ( "Node_number" )
22 $      Lits3     = ""
23 $      Lits6     = ""
24 $      WRITE Data_file "characteristic /noconsole"
25 $      WRITE Data_file "program TIME_3 /debug /argument=( ", -
26 $      Lits6, ",", Lits6, ",", Lits3, area, ".", node_num, Lits3, " )"
27 $      WRITE Data_file "program TODAY /norun /debug"
28 $
29 $ Close_file:
30 $      CLOSE Data_file
31 $      PURGE /NOLOG TIME_3.DAT
32 $      EXIT

```

The .DAT file used by EBUILD looked like this when EBUILD ran on node number 1.241

```

1      characteristic /noconsole
2      program TIME_3 /debug /argument=( "", "", ""1.241"" )
3      program TODAY /norun /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from TIME_3:

```
SH SYS
! Available: Pages: 17825, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:11.47 Total:  0 00:00:11.63
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program TIME_3, priority 16 is waiting.
!
GO
! Job 5, process 1, program TIME_3 running.
!
! 1) About to connect with VMS node: 1.241...
!
! 2) Successful connection with VMS node
!
! 3) About to set system time...
!
! 4) Creating job to run VAX FORTRAN program
!
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]TODAY.EXE;2
!
! Job 6, process 1, program TODAY needs attention.
! Module TODAY$MAIN
! 18:
!>>19: IMPLICIT NONE
! 20:
! 21: INTEGER          Month_length(12) /31, 28, 31, 30, 31, 30,
! 22: 1                  31, 31, 30, 31, 30, 31/
! 23:
! Job 5, process 1, program TIME_3 has exited.
!
GO
! Job 6, process 1, program TODAY running.
!
!
!      14:41:17 Monday      2nd March      1987  W  9, Y  61, C 31472
!
!
! Job 6, process 1, program TODAY has exited.
!
EXIT
```

D.5 EXAMPLE OF USING VAXELN INTERNAL TIME VALUES - TIME_4.PAS

This program reads VAXELN internal time values and manipulates them

```

1  {-----}
2  SOURCE:      TIME_4.PAS
3
4  PURPOSE:     To demonstrate access of system symbols for VAXELN
5                system time values.
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG TIME_4
8
9  LINK:        $ LINK /DEBUG TIME_4, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ ELN$:RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT TIME_4
14
15 NOTES:       1) Command procedure TIME_4.COM compiles, links
16                and builds this module into a VAXELN system
17 -----}
18 MODULE Time_4 [IDENT ('V1.000')];
19
20 CONST
21     Nov_17_1858          = 0;    { earliest time }
22
23 VAR
24     System_time_set      : BOOLEAN;
25
26     Start_time,
27     Idle_time,
28     Previous_job_time    : LARGE_INTEGER;

```


EXAMPLES OF SYNCHRONIZATION AND TIME

```

29
30     KER$GB_TIME_SET           : [EXTERNAL] BOOLEAN;
31
32     KER$GQ_SYSTEM_TIME,
33     KER$GQ_START_TIME,
34     KER$GQ_IDLE_TIME,
35     KER$GQ_PREV_JOB_TIME      : [EXTERNAL] LARGE_INTEGER;
36
37
38 {----- PROGRAM BLOCK -----}
39
40 PROGRAM Time_4 ( INPUT, OUTPUT );
41
42 BEGIN
43
44 { Initialize variables...
45   NOTE: KER$GQ_IDLE_TIME & KER$GQ_PREV_JOB_TIME are held as absolute
46   times starting at 17-Nov-1858 00:00:00.00.
47
48   When the time has been set:
49       these must be subtracted from 17-Nov-1858 00:00:00.00
50       to generate a delta time (bit 63 set)
51
52   When the time is not set:
53       the clock starts at 17-Nov-1858 00:00:00.00
54 }
55
56 System_time_set    := KER$GB_TIME_SET;
57 Start_time         := KER$GQ_START_TIME;
58
59 CASE System_time_set OF
60
61   TRUE : BEGIN
62       WRITELN ( 'System time has been set explicitly...' );
63       Idle_time      := Nov_17_1858 - KER$GQ_IDLE_TIME;
64       Previous_job_time := Nov_17_1858 - KER$GQ_PREV_JOB_TIME;
65       END;
66
67   FALSE : BEGIN
68       Idle_time      := Start_time - KER$GQ_IDLE_TIME;
69       Previous_job_time := Start_time - KER$GQ_PREV_JOB_TIME;
70       END;
71 END {CASE};

```

```

72
73      WRITELN ( 'System time: ',
74                TIME_STRING ( KER$GQ_SYSTEM_TIME ));
75      WRITELN ( 'Start time: ',
76                TIME_STRING ( Start_time ));
77      WRITELN ( 'Idle time: ',
78                TIME_STRING ( Idle_time ));
79      WRITELN ( 'Previous job time: ',
80                TIME_STRING ( Previous_job_time ));
81
82  END {of PROGRAM}.
83  END {of MODULE};

```

D.5.1 Running TIME_4

Use the command procedure of the same name to build a system with TIME_4:

```

1  $      ! TIME_4.COM
2  $      ! Command procedure to compile and link the EPASCAL module
3  $      ! TIME_4
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $      !
8  $      Compile_Time_4:
9  $          @ELN_COMPILE_1 TIME_4
10 $      Link_Time_4:
11 $          @ELN_LINK_1 TIME_4
12 $      System_build:
13 $          @ELN_EBUILD_1 TIME_4
14 $      Switch_off_verify:
15 $          SET NOVERIFY
16 $          EXIT

```

The .DAT file looks like this:

```

1      characteristic /noconsole /nonetwork /nofile /noserver
2      program TIME_4 /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from TIME_4 when a user sets the time:

```
SET TIME '17-MAR-1987 14:37:05.00'
SH SYS
! Available: Pages: 1512, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:00.02 Total:  0 00:00:00.04
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program TIME_4, priority 16 is waiting.
!
GO
! Job 4, process 1, program TIME_4 running.
!System time has been set explicitly...
!System time: 17-MAR-1987 14:37:06.11
!Start time:  17-MAR-1987 14:37:05.00
!Idle time:      0 00:00:00.94
!Previous job time:  0 00:00:00.02
! Job 4, process 1, program TIME_4 has exited.
!
EXIT
```

Output from TIME_4 when the time is NOT set:

```
SH SYS
! Available: Pages: 1512, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:15.95 Total:  0 00:00:16.07
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program TIME_4, priority 16 is waiting.
!
GO
! Job 4, process 1, program TIME_4 running.
!System time: 17-NOV-1858 00:00:17.83
!Start time:  17-NOV-1858 00:00:00.00
!Idle time:      0 00:00:17.58
!Previous job time:  0 00:00:00.02
! Job 4, process 1, program TIME_4 has exited.
!
EXIT
```

D.6 EXAMPLE OF USING WAIT ROUTINE - SYNCH_1.PAS

This program simply delays itself by calling WAIT_ANY without specifying an object but using the TIME argument.

```

1  {
2  SOURCE:          SYNCH_1.PAS
3
4  PURPOSE:         Demonstrates use of WAIT_ANY to delay for a period
5                    of seconds using the timeout facility of the WAIT_ANY
6                    procedure
7
8  COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_1
9
10 LINK:            $ LINK /DEBUG SYNCH_1, -
11                  _$ ELN$:RTLSHARE /LIBRARY, -
12                  _$ RTL /LIBRARY
13
14 BUILD:           $ EBUILD /NOEDIT SYNCH_1
15
16 NOTES:           1) Command procedure SYNCH_1.COM compiles, links and
17                   builds this module into a system
18
19 }
20 MODULE Synch_1 [IDENT ('V1.000')];
21
22 CONST
23     Interval = '0000 00:00:05.00';      { 5 secs }
24
25 VAR
26     Wait_result,
27     Status_returned,
28     Status_from_wait : INTEGER;
29
30     Current_time,
31     Binary_interval  : LARGE_INTEGER;
32
33 { ----- }
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

34
35 PROCEDURE Print_time;
36
37 BEGIN
38     GET_TIME( Current_time,
39               STATUS := Status_returned );
40     IF NOT ODD ( Status_returned ) THEN
41         WRITELN ( 'GET_TIME status was: ', Status_returned :1 );
42
43     WRITELN ( 'The time now is: ', TIME_STRING (Current_time) );
44 END;
45
46 { ----- }
47
48 PROGRAM Synch_1 (OUTPUT);
49
50 BEGIN
51     Binary_interval := TIME_VALUE ( Interval );
52
53     Print_time;
54     WRITELN ( 'About to wait..... ' );
55
56     WAIT_ANY ( RESULT := Wait_result,
57               TIME     := Binary_interval,
58               STATUS := Status_from_wait );
59     IF NOT ODD ( Status_from_wait ) THEN
60         WRITELN ( 'WAIT status was: ', Status_from_wait:1);
61
62     WRITELN ( 'Wait completed' );
63     WRITE   ( 'Value of wait_result was: ', Wait_result:1 );
64
65     IF Wait_result = 0 THEN
66         WRITELN ( ' indicating timeout.' )
67     ELSE
68         WRITELN;
69
70     Print_time;
71
72 END { of PROGRAM }.
73 END { of MODULE };

```

D.6.1 Running SYNCH_1

Use the command procedure of the same name to build a system with SYNCH_1:

```

1 $      ! SYNCH_1.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module SYNCH_1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1 SYNCH_1
10 $ Link:
11 $          @ELN_LINK_1 SYNCH_1
12 $ System_build:
13 $          @ELN_EBUILD_1 SYNCH_1
14 $ Switch_off_verify:
15 $      SET NOVERIFY
16 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /nofile /noserver
2      program SYNCH_1 /debug

```

Output from SYNCH_1:

```

SET TIME '02-MAR-1987 14:55:55.00'
SH SYS
! Available: Pages: 17890, Page table slots: 53, Pool blocks: 283
! Time since SET TIME: Idle:  0 00:00:03.03 Total:  0 00:00:03.08
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_1, priority 16 is waiting.
!
GO
! Job 4, process 1, program SYNCH_1 running.
!The time now is:  2-MAR-1987 14:56:01.21
!About to wait.....
!Wait completed
!Value of wait_result was: 0 indicating timeout.
!The time now is:  2-MAR-1987 14:56:07.48
! Job 4, process 1, program SYNCH_1 has exited.
!
EXIT

```

EXAMPLES OF SYNCHRONIZATION AND TIME

D.7 EXAMPLE OF WAITING FOR AN EVENT - SYNCH_2.PAS

This program waits for an EVENT to be signalled from a subprocess that delays itself by 5 seconds using the WAIT timeout facility

```
1 {
2 SOURCE:          SYNCH_2.PAS
3
4 PURPOSE:         Demonstrates waiting for an event
5
6 COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_2
7
8 LINK:            $ LINK /DEBUG SYNCH_2, -
9                  _$ ELN$:RTLSHARE /LIBRARY, -
10                 _$ RTL /LIBRARY
11
12 BUILD:           $ EBUILD /NOEDIT SYNCH_2
13
14 NOTES:           1) Command procedure SYNCH_2.COM compiles, links and
15                  builds this module into a system
16
17 }
18 MODULE Synch_2 [IDENT ('V1.000')];
19
20 CONST
21     Interval = '0000 00:00:05.00';    { 5 secs }
22
23 VAR
24     Process_ID      : PROCESS;
25
26     Binary_interval : LARGE_INTEGER;
27
28     Wait_result,
29     Status_returned,
30     Status_from_signal,
31     Status_from_wait : INTEGER;
32
33
34 { ----- }
35
```

```

36 PROCESS_BLOCK Flag_setter ( Flag_to_signal : EVENT );
37
38 BEGIN
39     Binary_interval := TIME_VALUE ( Interval );
40
41     WAIT_ANY (      RESULT := Wait_result,
42                TIME     := Binary_interval,
43                STATUS  := Status_from_wait );
44     IF NOT ODD ( Status_from_wait ) THEN
45         WRITELN ( 'WAIT status was: ', Status_from_wait:1);
46
47     WRITELN ( 'In subprocess about to signal event... ');
48
49     SIGNAL ( Flag_to_signal,
50             STATUS := Status_from_signal );
51     IF NOT ODD ( Status_from_signal ) THEN
52         WRITELN ( 'SIGNAL status was: ', Status_from_signal:1 );
53
54 END;
55 { ----- }
56 PROGRAM Synch_2 (OUTPUT);
57
58 VAR
59     Event_flag      : EVENT;
60
61 BEGIN
62     CREATE_EVENT (    Event_flag,
63                     EVENT$CLEARED,
64                     STATUS := Status_returned );
65     IF NOT ODD ( Status_returned ) THEN
66         WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);
67
68     CREATE_PROCESS (  Process_id,
69                     Flag_setter,
70                     Event_flag,
71                     STATUS := Status_returned );
72     IF NOT ODD ( Status_returned ) THEN
73         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
74
75     WRITELN ( 'Master process about to wait..... ' );
76
77     WAIT_ANY      (    Event_flag,
78                     RESULT := Wait_result,
79                     STATUS := Status_from_wait );
80     IF NOT ODD ( Status_from_wait ) THEN
81         WRITELN ( 'WAIT status was: ', Status_from_wait:1);
82
83     WRITELN ( 'Wait completed' );
84     WRITE   ( 'Value of wait_result was: ', Wait_result:1 );
85
86 END { of PROGRAM }.
87 END { of MODULE };

```


EXAMPLES OF SYNCHRONIZATION AND TIME

D.7.1 Running SYNCH_2

Use the command procedure of the same name to build a system with SYNCH_2:

```
1 $      ! SYNCH_2.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module SYNCH_2
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1 SYNCH_2
10 $ Link:
11 $          @ELN_LINK_1 SYNCH_2
12 $ System_build:
13 $          @ELN_EBUILD_1 SYNCH_2
14 $ Switch_off_verify:
15 $          SET NOVERIFY
16 $          EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole /nofile /noserver
2      program SYNCH_2 /debug
```

Output from SYNCH_2:

```

SET TIME '02-MAR-1987 14:57:30.00'
SH SYS
! Available: Pages: 17888, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:01.67 Total:  0 00:00:01.71
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_2, priority 16 is waiting.
!
GO
! Job 4, process 1, program SYNCH_2 running.
!Master process about to wait.....
! Job 4, process 2, program SYNCH_2 needs attention.
! Module SYNCH_2
! 38:
!>>39: BEGIN
! 40:   Binary_interval := TIME_VALUE ( Interval );
! 41:
! 42: WAIT_ANY (      RESULT := Wait_result,
! 43:   TIME      := Binary_interval,
!
GO
! Job 4, process 2, program SYNCH_2 running.
!In subprocess about to signal event...
! Job 4, process 2, program SYNCH_2 has exited.
!
! Job 4, process 1, program SYNCH_2 running.
!Wait completed
!Value of wait_result was: 1
! Job 4, process 1, program SYNCH_2 has exited.
!
EXIT

```

EXAMPLES OF SYNCHRONIZATION AND TIME

D.8 EXAMPLE OF WAITING FOR OBJECTS - SYNCH_3.PAS

This program waits for an EVENT and three other objects. The EVENT is signalled from a subprocess that delays itself by 5 seconds using the WAIT timeout facility

```
1  {
2  SOURCE:          SYNCH_3.PAS
3
4  PURPOSE:         Demonstrates waiting for several objects
5                   Three of the objects are dummies just for
6                   demonstration purposes.
7
8  COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_3
9
10 LINK:            $ LINK /DEBUG SYNCH_3, -
11                  _$ ELN$:RTLSHARE /LIBRARY, -
12                  _$ RTL /LIBRARY
13
14 BUILD:           $ EBUILD /NOEDIT SYNCH_3
15
16 NOTES:           1) Command procedure SYNCH_3.COM compiles, links and
17                  builds this module into a system
18
19 }
20 MODULE Synch_3 [IDENT ('V1.000')];
21
22 CONST
23     Interval          = '0000 00:00:05.00';    { 5 secs }
24     No_of_dummy_events = 3;
25
26 VAR
27     Process_ID        : PROCESS;
28     Dummy_event        : ARRAY [1..No_of_dummy_events] OF EVENT;
29     Event_flag         : EVENT;
30
31     Binary_interval    : LARGE_INTEGER;
32
33     Wait_result,
34     Status_returned,
35     Status_from_signal,
36     Status_from_wait   : INTEGER;
37
38
39 { ----- }
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

40
41 PROCESS_BLOCK Flag_setter ( Flag_to_signal : EVENT );
42
43 BEGIN
44     Binary_interval := TIME_VALUE ( Interval );
45
46     { wait for some seconds before signalling the event }
47
48     WAIT_ANY (      RESULT := Wait_result,
49                  TIME    := Binary_interval,
50                  STATUS  := Status_from_wait );
51     IF NOT ODD ( Status_from_wait ) THEN
52         WRITELN ( 'WAIT status was: ', Status_from_wait:1);
53
54     WRITELN ( 'In subprocess about to signal event... ');
55
56     SIGNAL ( Flag_to_signal,
57             STATUS := Status_from_signal );
58     IF NOT ODD ( Status_from_signal ) THEN
59         WRITELN ( 'SIGNAL status was: ', Status_from_signal:1 );
60
61 END;
62
63 { ----- }
64
65 PROGRAM Synch_3 (OUTPUT);
66
67 VAR
68     Event_count : 1..No_of_dummy_events;
69
70 BEGIN
71     CREATE_EVENT (      Event_flag,
72                      EVENT$CLEARED,
73                      STATUS := Status_returned );
74     IF NOT ODD ( Status_returned ) THEN
75         WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);
76
77     { create three dummy objects - in this case EVENT objects }
78
79     FOR Event_count := 1 TO No_of_dummy_events DO
80         BEGIN
81             CREATE_EVENT ( Dummy_event[Event_count],
82                           EVENT$CLEARED,
83                           STATUS := Status_returned );
84             IF NOT ODD ( Status_returned ) THEN
85                 WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);
86         END;
87

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```
88  CREATE_PROCESS ( Process_id,  
89                  Flag_setter,  
90                  Event_flag,  
91                  STATUS := Status_returned );  
92  IF NOT ODD ( Status_returned ) THEN  
93      WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);  
94  
95      WRITELN ( 'Master process about to wait..... ' );  
96  
97  { wait for any object to be signalled }  
98  
99  WAIT_ANY      ( Dummy_event[1],  
100                Dummy_event[2],  
101                Event_flag,  
102                Dummy_event[3],  
103                RESULT := Wait_result,  
104                STATUS := Status_from_wait );  
105  IF NOT ODD ( Status_from_wait ) THEN  
106      WRITELN ( 'WAIT status was: ', Status_from_wait:1);  
107  
108      WRITELN ( 'Wait completed' );  
109      WRITE   ( 'Value of wait_result was: ', Wait_result:1 );  
110  
111  END { of PROGRAM }.  
112  END { of MODULE };
```

D.8.1 Running SYNCH_3

Use the command procedure of the same name to build a system with SYNCH_3:

```

1 $      ! SYNCH_3.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module SYNCH_3
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1 SYNCH_3
10 $ Link:
11 $          @ELN_LINK_1 SYNCH_3
12 $ System_build:
13 $          @ELN_EBUILD_1 SYNCH_3
14 $ Switch_off_verify:
15 $      SET NOVERIFY
16 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /nofile /noserver
2      program SYNCH_3 /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from SYNCH_3:

```
SH SYS
! Available: Pages: 17889, Page table slots: 53, Pool blocks: 282
! Time since SET TIME: Idle:  0 00:00:49.48 Total:  0 00:00:49.61
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_3, priority 16 is waiting.
!
GO
! Job 4, process 1, program SYNCH_3 running.
!Master process about to wait.....
! Job 4, process 2, program SYNCH_3 needs attention.
! Module SYNCH_3
! 42:
!>>43: BEGIN
! 44:     Binary_interval := TIME_VALUE ( Interval );
! 45:
! 46: { wait for some seconds before signalling the event }
! 47:
!
GO
! Job 4, process 2, program SYNCH_3 running.
!In subprocess about to signal event...
! Job 4, process 2, program SYNCH_3 has exited.
!
! Job 4, process 1, program SYNCH_3 running.
!Wait completed
!Value of wait_result was: 3
! Job 4, process 1, program SYNCH_3 has exited.
!
EXIT
```

D.9 EXAMPLE OF WAITING FOR A PROCESS - SYNCH_4.PAS

This program waits for a PROCESS. A wait for a process expires when a PROCESS exits either by running out of code, calling EXIT or being DELETED.

After being created this subprocess is held, waiting for a flag to be signalled from the master process.

```

1  {
2  SOURCE:          SYNCH_4.PAS
3
4  PURPOSE:         Demonstrates waiting for a process object
5
6  COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_4
7
8  LINK:            $ LINK /DEBUG SYNCH_4, -
9                  _$ ELN$:RTLSHARE /LIBRARY, -
10                 _$ RTL /LIBRARY
11
12 BUILD:           $ EBUILD /NOEDIT SYNCH_4
13
14 NOTES:           1) Command procedure SYNCH_4.COM compiles, links and
15                  builds this module into a system
16
17 }
18 MODULE Synch_4 [IDENT ('V1.000')];
19
20 VAR
21     Event_flag      : EVENT;
22     Subprocess_id    : PROCESS;
23
24     Wait_result,
25     Status_returned,
26     Status_from_wait : INTEGER;
27
28 { ----- }
29
30 PROCESS_BLOCK Time_waster ( Flag : EVENT );
31
32 VAR
33     I,
34     J,
35     Ret_stat        : INTEGER;
36
37 BEGIN
38

```


EXAMPLES OF SYNCHRONIZATION AND TIME

```

39 { process waits here for the master process to signal the event thus
40     satisfying the wait and setting the subprocess on its way }
41
42     WAIT_ANY (    Flag,
43                 STATUS := Ret_stat );
44     IF NOT ODD ( Ret_stat ) THEN
45         WRITELN ( 'WAIT status was: ', Ret_stat:1 );
46
47 { just waste some time }
48
49     FOR I := 1 TO 10000 DO
50         J := I;
51     END;
52
53 { ----- }
54
55 PROGRAM Synch_4 (OUTPUT);
56
57 BEGIN
58
59 { create the event that will hold the subprocess after its created }
60
61     CREATE_EVENT (    Event_flag,
62                     EVENT$CLEARED,
63                     STATUS := Status_returned );
64     IF NOT ODD ( Status_returned ) THEN
65         WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);
66
67     CREATE_PROCESS (    Subprocess_id,
68                       Time_waster,
69                       Event_flag,
70                       STATUS := Status_returned );
71     IF NOT ODD ( Status_returned ) THEN
72         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
73
74     WRITELN ( 'Master process about to wait..... ' );
75
76 { let the subprocess go by 'dropping' the flag }
77
78     SIGNAL ( Event_flag,
79             STATUS := Status_returned );
80     IF NOT ODD ( Status_returned ) THEN
81         WRITELN ( 'SIGNAL status was: ', Status_returned :1);

```

```

82
83 { wait for the subprocess to expire }
84
85 WAIT_ANY      ( Subprocess_id,
86                  RESULT := Wait_result,
87                  STATUS := Status_from_wait );
88 IF NOT ODD ( Status_from_wait ) THEN
89     WRITELN ( 'WAIT status was: ', Status_from_wait:1);
90
91     WRITELN ( 'Wait completed' );
92     WRITE   ( 'Value of wait_result was: ', Wait_result:1 );
93
94 END { of PROGRAM }.
95 END { of MODULE };

```

D.9.1 Running SYNCH_4

Use the command procedure of the same name to build a system with SYNCH_4:

```

1 $      ! SYNCH_4.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module SYNCH_4
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1 SYNCH_4
10 $ Link:
11 $          @ELN_LINK_1 SYNCH_4
12 $ System_build:
13 $          @ELN_EBUILD_1 SYNCH_4
14 $ Switch_off_verify:
15 $          SET NOVERIFY
16 $          EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /nofile /noserver
2      program SYNCH_4 /debug

```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from SYNCH_4:

```
GO
! Job 4, process 1, program SYNCH_4 running.
!Master process about to wait.....
! Job 4, process 2, program SYNCH_4 needs attention.
! Module SYNCH_4
! 36:
!>>37: BEGIN
! 38:
! 39: { process waits here for the master process to signal the event thus
! 40: satisfying the wait and setting the subprocess on its way }
! 41:
SH SYS
! Available: Pages: 17842, Page table slots: 51, Pool blocks: 264
! Time since SET TIME: Idle: 0 00:00:17.14 Total: 0 00:00:17.42
! Time used by past jobs: 0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_4, priority 16 is waiting.
!
GO
! Job 4, process 2, program SYNCH_4 running.
! Job 4, process 2, program SYNCH_4 has exited.
!
! Job 4, process 1, program SYNCH_4 running.
!Wait completed
!Value of wait_result was: 1
! Job 4, process 1, program SYNCH_4 has exited.
!
EXIT
```

D.10 EXAMPLE OF MUTEX - SYNCH_5.PAS

This program uses a MUTEX for controlling access to an array.

```

1  {
2  SOURCE:          SYNCH_5.PAS
3
4  PURPOSE:         Demonstrates use of a MUTEX to control access to an
5                    array
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_5, ELN$:RTLOBJECT /LIB
8
9  LINK:            $ LINK /DEBUG SYNCH_5, -
10                  _$ ELN$:RTLSHARE /LIBRARY, -
11                  _$ RTL /LIBRARY
12
13 BUILD:           $ EBUILD /NOEDIT SYNCH_5
14
15 NOTES:           1) Command procedure SYNCH_5.COM compiles, links and
16                   builds this module into a system
17
18 }
19 MODULE Synch_5 [IDENT ('V1.000')];
20
21 INCLUDE
22     $MUTEX;          { contains mutex definitions }
23
24 CONST
25     LF               = '(10);    { line-feed }
26     Fill_limit       = 5;
27     Limit            = 100;
28
29 VAR
30     List_of_integers : ARRAY [1..Limit] OF INTEGER;
31     Access_control   : MUTEX;
32     Number_of_fills  : INTEGER := ZERO;
33     Status_returned  : INTEGER;
34 { ----- }
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

35
36 PROCESS_BLOCK Array_filler;
37
38 VAR
39     I : INTEGER;
40
41 BEGIN
42     WHILE Number_of_fills < Fill_limit DO
43         BEGIN
44
45             { lock the mutex created by the master process - this increments
46               the count for the mutex. When the master process 'unlocks', the
47               count is decremented and at most one process - this one - can
48               proceed
49             }
50             LOCK_MUTEX ( Access_control );
51
52             FOR I := 1 TO Limit DO List_of_integers[I] :=
53                                     List_of_integers[I] + I * 2;
54
55             Number_of_fills := Number_of_fills + 1;
56
57             WRITELN ( LF, 'Fill number ', Number_of_fills:1,
58                     ' completed, contents of array --> ', LF);
59
60             { now let the next waiting process gain control of the mutex
61               - 'Array_writer' }
62
63             UNLOCK_MUTEX ( Access_control );
64         END;
65     END;
66
67     { ----- }
68
69 PROCESS_BLOCK Array_writer;
70
71 VAR
72     I : 1..Limit;
73
74 BEGIN
75     WHILE Number_of_fills < Fill_limit DO
76         BEGIN
77
78             { stop other processes using the array by locking the mutex }
79
80             LOCK_MUTEX ( Access_control );
81             FOR I := 1 TO Limit DO
82                 BEGIN
83                     IF ( (I-1) MOD 16 = 0 ) THEN WRITELN;
84                     WRITE ( List_of_integers[I]:5 );
85                 END;
86             WRITELN;
87

```

```

88 { decrement the mutex counter - allow the next waiting process,
89   if any, to continue }
90
91   UNLOCK_MUTEX ( Access_control );
92   END;
93 END;
94
95 { ----- }
96
97 PROGRAM Synch_5 (OUTPUT);
98
99 VAR
100     Filler_ID,
101     Writer_ID      : PROCESS;
102
103 BEGIN
104
105 { creating the mutex initializes its counter to -1 }
106
107 CREATE_MUTEX ( Access_control,
108               Status_returned );
109     IF NOT ODD ( Status_returned ) THEN
110         WRITELN ( 'CREATE_MUTEX status was: ', Status_returned :1);
111
112 { this master process now has control of the mutex and increments
113   the mutex counter }
114
115 LOCK_MUTEX ( Access_control );
116
117 { create two processes that will manipulate the array }
118
119 CREATE_PROCESS ( Filler_id,
120                 Array_filler,
121                 STATUS := Status_returned );
122     IF NOT ODD ( Status_returned ) THEN
123         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
124
125 CREATE_PROCESS ( Writer_id,
126                 Array_writer,
127                 STATUS := Status_returned );
128     IF NOT ODD ( Status_returned ) THEN
129         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
130
131 { This ensures we don't miss the first or second write operations }
132
133     WRITELN ( 'Please hit the return key to start' );
134     READLN;
135

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```
136 { let the first process go - Array_filler }
137
138 UNLOCK_MUTEX ( Access_control );
139
140 { wait for the two processes to complete their operations and exit }
141
142 WAIT_ALL ( Filler_ID,
143           Writer_ID,
144           STATUS := Status_returned );
145           IF NOT ODD ( Status_returned ) THEN
146             WRITELN ( 'WAIT status was: ', Status_returned:1);
147
148 END { of PROGRAM }.
149 END { of MODULE };
```

D.10.1 Running SYNCH_5

Use the command procedure of the same name to build a system with SYNCH_5:

```
1 $      ! SYNCH_5.COM
2 $      ! Command procedure to compile and link the VAXELN
3 $      ! module SYNCH_5
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG SYNCH_5, ELN$:RTLOBJECT /LIBRARY
13 $      SET NOVERIFY
14 $ Link:
15 $      @ELN_LINK_1 SYNCH_5
16 $ System_build:
17 $      @ELN_EBUILD_1 SYNCH_5
18 $ Switch_off_verify:
19 $      SET NOVERIFY
20 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole /nofile /noserver
2      program SYNCH_5 /debug
```

EXAMPLES OF SYNCHRONIZATION AND TIME

If, when using EDEBUD, one does not issue the CANCEL CONTROL command the output from SYNCH_5 looks like this:

```
GO
! Job 4, process 1, program SYNCH_5 running.
!Please hit the return key to start

! Job 4, process 2, program SYNCH_5 needs attention.
! Module SYNCH_5
! 40:
!>>41: BEGIN
! 42: WHILE Number_of_fills < Fill_limit DO
! 43: BEGIN
! 44:
! 45: { lock the mutex created by the master process - this increments
! Job 4, process 3, program SYNCH_5 needs attention.
! Module SYNCH_5
! 73:
!>>74: BEGIN
! 75: WHILE Number_of_fills < Fill_limit DO
! 76: BEGIN
! 77:
! 78: { stop other processes using the array by locking the mutex }
!
GO
! Job 4, process 2, program SYNCH_5 running.
!
!Fill number 1 completed, contents of array -->
!
!Fill number 2 completed, contents of array -->
!
!Fill number 3 completed, contents of array -->
!
!Fill number 4 completed, contents of array -->
!
!Fill number 5 completed, contents of array -->
! Job 4, process 2, program SYNCH_5 has exited.
!
! Job 4, process 1, program SYNCH_5 running.
EXIT
```

Clearly the output has been lost.

EXAMPLES OF SYNCHRONIZATION AND TIME

Issuing CANCEL CONTROL allows the processes to run automatically without operator intervention thus:

CANCEL CONTROL

SH SYS

```
! Available: Pages: 17890, Page table slots: 53, Pool blocks: 283
! Time since SET TIME: Idle:  0 00:01:06.54 Total:  0 00:01:06.70
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_5, priority 16 is waiting.
!
```

GO

```
! Job 4, process 1, program SYNCH_5 running.
! Please hit the return key to start
!
```

! Fill number 1 completed, contents of array -->

```
!
!   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32
!  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64
!  66  68  70  72  74  76  78  80  82  84  86  88  90  92  94  96
!  98 100 102 104 106 108 110 112 114 116 118 120 122 124 126 128
! 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158 160
! 162 164 166 168 170 172 174 176 178 180 182 184 186 188 190 192
! 194 196 198 200
!
```

! Fill number 2 completed, contents of array -->

```
!
!   4   8  12  16  20  24  28  32  36  40  44  48  52  56  60  64
!  68  72  76  80  84  88  92  96 100 104 108 112 116 120 124 128
! 132 136 140 144 148 152 156 160 164 168 172 176 180 184 188 192
! 196 200 204 208 212 216 220 224 228 232 236 240 244 248 252 256
! 260 264 268 272 276 280 284 288 292 296 300 304 308 312 316 320
! 324 328 332 336 340 344 348 352 356 360 364 368 372 376 380 384
! 388 392 396 400
!
```

! Fill number 3 completed, contents of array -->

```
!
!   6  12  18  24  30  36  42  48  54  60  66  72  78  84  90  96
! 102 108 114 120 126 132 138 144 150 156 162 168 174 180 186 192
! 198 204 210 216 222 228 234 240 246 252 258 264 270 276 282 288
! 294 300 306 312 318 324 330 336 342 348 354 360 366 372 378 384
! 390 396 402 408 414 420 426 432 438 444 450 456 462 468 474 480
! 486 492 498 504 510 516 522 528 534 540 546 552 558 564 570 576
! 582 588 594 600
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

!
!Fill number 4 completed, contents of array -->
!
! 8   16   24   32   40   48   56   64   72   80   88   96  104  112  120  128
! 136  144  152  160  168  176  184  192  200  208  216  224  232  240  248  256
! 264  272  280  288  296  304  312  320  328  336  344  352  360  368  376  384
! 392  400  408  416  424  432  440  448  456  464  472  480  488  496  504  512
! 520  528  536  544  552  560  568  576  584  592  600  608  616  624  632  640
! 648  656  664  672  680  688  696  704  712  720  728  736  744  752  760  768
! 776  784  792  800
!
!Fill number 5 completed, contents of array -->
!
! 10   20   30   40   50   60   70   80   90  100  110  120  130  140  150  160
! 170  180  190  200  210  220  230  240  250  260  270  280  290  300  310  320
! 330  340  350  360  370  380  390  400  410  420  430  440  450  460  470  480
! 490  500  510  520  530  540  550  560  570  580  590  600  610  620  630  640
! 650  660  670  680  690  700  710  720  730  740  750  760  770  780  790  800
! 810  820  830  840  850  860  870  880  890  900  910  920  930  940  950  960
! 970  980  990 1000
! Job 4, process 1, program SYNCH_5 has exited.
!
EXIT

```

EXAMPLES OF SYNCHRONIZATION AND TIME

D.11 EXAMPLE OF MUTEX - SYNCH_6.PAS

This program creates a large number of processes and uses a mutex to allow each in turn to write to the terminal. The EBUILD requires that the number of page slots be increased as well as the number of pool blocks - please see the .DAT file later

```
1  {
2  SOURCE:          SYNCH_6.PAS
3
4  PURPOSE:         Demonstrates multiple processes synchronizing
5                    using a mutex
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_6
8
9  LINK:            $ LINK /DEBUG SYNCH_6, -
10                  _$ ELN$:RTLSHARE /LIBRARY, -
11                  _$ RTL /LIBRARY
12
13 BUILD:           $ EBUILD /NOEDIT SYNCH_6
14
15 NOTES:           1) Command procedure SYNCH_6.COM compiles, links and
16                   builds this module into a system
17
18 }
19 MODULE Synch_6 [IDENT ('V1.000')];
20
21 INCLUDE
22     $MUTEX;
23
24 CONST
25     Process_limit    = 100;
26
27 VAR
28     Process_IDs      : ARRAY [1..Process_limit] OF PROCESS;
29     Process_lock      : MUTEX;
30     Status_returned   : INTEGER;
31
32 { ----- }
33
```

EXAMPLES OF SYNCHRONIZATION AND TIME

```

34  PROCEDURE Write_time_now;
35
36  VAR
37      Now : LARGE_INTEGER;
38
39  BEGIN
40      GET_TIME ( Now );    { no status checks for speed }
41      Writeln ( TIME_STRING ( Now ) );
42  END;
43
44  { ----- }
45
46  PROCESS_BLOCK Subprocess ( K : INTEGER );
47
48  BEGIN
49
50  LOCK_MUTEX ( Process_lock );
51
52      WRITE ( K : 4 );
53
54      IF ( K MOD 20 = 0 ) THEN Writeln;
55
56  UNLOCK_MUTEX ( Process_lock );
57
58  END;
59
60  { ----- }
61
62  PROGRAM Synch_6 ( OUTPUT );
63
64  VAR
65      I          : 1..Process_limit;
66
67  BEGIN
68
69  CREATE_MUTEX ( Process_lock );
70  LOCK_MUTEX   ( Process_lock );
71
72  WRITE ( 'About to create ', Process_limit:1, ' processes... ');
73  Write_time_now;
74
75  FOR I := 1 TO Process_limit DO
76      BEGIN
77          CREATE_PROCESS ( Process_IDs [I],
78                          Subprocess,
79                          I,
80                          STATUS := Status_returned );
81          IF NOT ODD ( Status_returned ) THEN
82              Writeln ( 'Bad status returned by CREATE_PROCESS: ',
83                      Status_returned:1 );
84      END;

```

EXAMPLES OF SYNCHRONIZATION AND TIME

```
85
86 WRITE ( '      ...process creation complete ');
87 Write_time_now;
88
89 WRITELN ( 'Please hit return to start... ');
90 READLN;
91
92 WRITELN ( ' ':10, '----- SUBPROCESS OUTPUT -----' );
93 WRITELN;
94
95 UNLOCK_MUTEX ( Process_lock );
96
97 WAIT_ANY ( Process_IDs [Process_limit] );
98
99 END {of PROGRAM}.
100 END {of MODULE};
```

D.11.1 Running SYNCH_6

Use the command procedure of the same name to build a system with SYNCH_6:

```
1 $      ! SYNCH_6.COM
2 $      ! Command procedure to compile and link the VAXELN
3 $      ! module SYNCH_6
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG SYNCH_6, ELN$:RTLOBJECT /LIBRARY
13 $      SET NOVERIFY
14 $ Link:
15 $      @ELN_LINK_1 SYNCH_6
16 $ System_build:
17 $      @ELN_EBUILD_1 SYNCH_6
18 $ Switch_off_verify:
19 $      SET NOVERIFY
20 $      EXIT
```

The .DAT files looks like this:

```
1      characteristic /noconsole /nonetwork /nofile /noserver -
2          /objects=800 /processes=120
3      program SYNCH_6 /debug
```

EXAMPLES OF SYNCHRONIZATION AND TIME

Output from SYNCH_6:

SET TIME '18-MAR-1987 11:33:55.00'

CANCEL CONTROL

SH SYSTEM

! Available: Pages: 1404, Page table slots: 109, Pool blocks: 698
! Time since SET TIME: Idle: 0 00:00:25.84 Total: 0 00:00:25.87
! Time used by past jobs: 0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_6, priority 16 is waiting.
!

GO

! Job 4, process 1, program SYNCH_6 running.
!About to create 100 processes... 18-MAR-1987 11:34:23.58
! ...process creation complete 18-MAR-1987 11:34:23.71
!Please hit return to start...

SHO SYSTEM

! Available: Pages: 520, Page table slots: 7, Pool blocks: 176
! Time since SET TIME: Idle: 0 00:00:33.18 Total: 0 00:00:33.77
! Time used by past jobs: 0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program SYNCH_6, priority 16 is waiting.
!

! Job 4, process 1, program SYNCH_6 running.

! ----- SUBPROCESS OUTPUT -----

! 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
! 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
! 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
! 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
! 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
! Job 4, process 1, program SYNCH_6 has exited.

EXIT

APPENDIX E
EXAMPLES OF COMMUNICATION TECHNIQUES

This appendix deals with examples using the VAXELN communication facilities.

E.1 EXAMPLE OF DATAGRAM COMMUNICATION - COMM_1.PAS

This program uses a datagram to communicate with its subprocess. Datagrams are not the preferred mechanism for communication, circuits are much more reliable.

```
1  {
2  SOURCE:      COMM_1.PAS
3
4  PURPOSE:     Demonstrates communication between a subprocess and
5                a master process using a datagram
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_1
8
9  LINK:        $ LINK /DEBUG COMM_1, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT COMM_1
14
15 NOTES:       1) Command procedure COMM_1.COM compiles, links and
16                builds this module into a system
17
18 }
```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

19  MODULE Comm_1 [IDENT ('V1.000')];
20
21  { define a message 'packet' }
22
23  TYPE
24      Message_packet = RECORD
25          Message_number : 0..MAXINT;
26          Timestring      : VARYING_STRING(23);
27      END;
28
29  VAR
30      Datagram          : ^Message_packet;
31      Jobs_port_value   : PORT;
32      Message_ID        : MESSAGE;
33      Status_returned   : INTEGER;
34      Sub_ID            : PROCESS;
35
36  { ----- }
37
38  PROCESS_BLOCK Sender;
39
40  VAR
41      Current_time_binary : LARGE_INTEGER;
42
43  BEGIN
44
45  GET_TIME ( Current_time_binary,
46              STATUS := Status_returned );
47      IF NOT ODD ( Status_returned ) THEN
48          WRITELN ( 'GET_TIME status was: ',
49                  Status_returned :1);
50
51  { complete the datagram's contents... }
52
53  WITH Datagram^ DO
54      BEGIN
55          Message_number := Datagram^.Message_number + 1;
56          Timestring      := TIME_STRING ( Current_time_binary );
57      END;
58
59  { ... then send it }
60
61  SEND ( Message_ID,
62          Jobs_port_value,
63          STATUS := Status_returned );
64      IF NOT ODD ( Status_returned ) THEN
65          WRITELN ( 'SEND status was: ',
66                  Status_returned :1);
67  END;
68
69  { ----- }

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

70
71
72 PROGRAM Comm_1 (OUTPUT);
73
74 BEGIN
75
76 { get a pointer to a datagram packet }
77
78 CREATE_MESSAGE ( Message_ID,
79                 Datagram,
80                 STATUS := Status_returned );
81     IF NOT ODD ( Status_returned ) THEN
82         WRITELN      ( 'CREATE_MESSAGE status was: ',
83                       Status_returned :1);
84
85 Datagram^.Message_number := 0;
86
87 { find the port value for the master process default port }
88
89 JOB_PORT ( Jobs_port_value,
90           STATUS := Status_returned );
91     IF NOT ODD ( Status_returned ) THEN
92         WRITELN      ( 'JOB_PORT status was: ',
93                       Status_returned :1);
94 { create the subprocess that's going to communicate with the
95   datagram
96 }
97
98 CREATE_PROCESS ( Sub_ID,
99                 Sender,
100                STATUS := Status_returned );
101     IF NOT ODD ( Status_returned ) THEN
102         WRITELN      ( 'CREATE_PROCESS status was: ',
103                       Status_returned :1);
104
105 { wait on the port i.e. wait for a message to arrive }
106
107 WAIT_ANY ( Jobs_port_value,
108           STATUS := Status_returned );
109     IF NOT ODD ( Status_returned ) THEN
110         WRITELN      ( 'WAIT status was: ',
111                       Status_returned :1);
112
113 RECEIVE ( Message_ID,
114          Datagram,
115          Jobs_port_value,
116          STATUS := Status_returned );
117     IF NOT ODD ( Status_returned ) THEN
118         WRITELN      ( 'RECEIVE status was: ',
119                       Status_returned :1);
120

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```
121 { read the message and display its contents }
122
123 WITH Datagram^ DO
124     BEGIN
125         WRITELN ( 'Contents of the message received: ' );
126         WRITELN ( 'Message number: ', Message_number:1 );
127         WRITELN ( 'Message text: ', Timestring );
128     END;
129
130 END { of PROGRAM }.
131 END { of MODULE };
```

E.1.1.1 Running COMM_1

Use the command procedure of the same name to build a system with COMM_1:

```

1 $      ! COMM_1.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module COMM_1
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1 COMM_1
10 $ Link:
11 $          @ELN_LINK_1    COMM_1
12 $ System_build:
13 $          @ELN_EBUILD_1  COMM_1
14 $ Switch_off_verify:
15 $          SET NOVERIFY
16 $          EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /nofile /noserver
2      program COMM_1 /debug

```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_1:

```
SH SYS
! Available: Pages: 17888, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:08.33 Total:  0 00:00:08.47
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_1, priority 16 is waiting.
!
GO
! Job 4, process 1, program COMM_1 running.
! Job 4, process 2, program COMM_1 needs attention.
! Module COMM_1
! 42:
!>>43: BEGIN
! 44:
! 45: GET_TIME ( Current_time_binary,
! 46:   STATUS := Status_returned );
! 47:   IF NOT ODD ( Status_returned ) THEN
GO
! Job 4, process 2, program COMM_1 running.
! Job 4, process 2, program COMM_1 has exited.
!
! Job 4, process 1, program COMM_1 running.
!Contents of the message received:
!Message number: 1
!Message text:  17-NOV-1858 00:00:13.82
! Job 4, process 1, program COMM_1 has exited.
!
EXIT
```

E.2 EXAMPLE OF DATAGRAM COMMUNICATION - COMM_2.PAS

Like COMM_1, this program uses a datagram to communicate with its subprocess. This time messages are sent in both directions. Datagrams are not the preferred mechanism for communication, circuits are much more reliable.

```

1  {
2  SOURCE:          COMM_2.PAS
3
4  PURPOSE:         Demonstrates communication between a subprocess and
5                    a master process using a datagram.
6                    This is an enhancement of COMM_1 and involves
7                    communication in two directions
8
9  COMPILE:         $ EPASCAL /LIST /DEBUG COMM_2
10
11 LINK:            $ LINK /DEBUG COMM_2, -
12                  _$ ELN$:RTLSHARE /LIBRARY, -
13                  _$ RTL /LIBRARY
14
15 BUILD:           $ EBUILD /NOEDIT COMM_2
16
17 NOTES:           1) Command procedure COMM_2.COM compiles, links and
18                  builds this module into a system
19
20 }
21 MODULE Comm_2 [IDENT ('V1.000')];
22
23 CONST
24     LF          = '(10);          { line-feed }
25
26 { the message packet (datagram) definition }
27
28 TYPE
29     Message_packet = RECORD
30         Message_number : 0..MAXINT;
31         Timestring      : VARYING_STRING(23);
32     END;
33
34 VAR
35     Datagram          : ^Message_packet;
36     Green_light        : EVENT;
37     Jobs_port_value    : PORT;
38     Message_ID         : MESSAGE;
39     Status_returned    : INTEGER;
40     Sub_ID              : PROCESS;
41
42 { ----- }

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

43
44 PROCEDURE Send_message;
45
46 BEGIN
47
48 SEND ( Message_ID,
49         Jobs_port_value,
50         STATUS := Status_returned );
51     IF NOT ODD ( Status_returned ) THEN
52     WRITELN      ( 'SEND status was: ',
53                   Status_returned :1);
54
55 END;
56
57 { ----- }
58
59 PROCEDURE Receive_message_and_display_it;
60
61 BEGIN
62
63 RECEIVE ( Message_ID,
64           Datagram,
65           Jobs_port_value,
66           STATUS := Status_returned );
67     IF NOT ODD ( Status_returned ) THEN
68     WRITELN      ( 'RECEIVE status was: ',
69                   Status_returned :1);
70
71 WITH Datagram^ DO
72     BEGIN
73         WRITELN ( 'Contents of the message received: ' );
74         WRITELN ( 'Message number: ', Message_number:1 );
75         WRITELN ( 'Message text:   ', Timestring );
76     END;
77
78 END;
79
80 { ----- }

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

81
82 PROCESS_BLOCK Sender;
83
84 VAR
85     Current_time_binary : LARGE_INTEGER;
86
87 BEGIN
88
89 GET_TIME ( Current_time_binary,
90             STATUS := Status_returned );
91     IF NOT ODD ( Status_returned ) THEN
92         WRITELN      ( 'GET_TIME status was: ',
93                        Status_returned :1);
94
95 WITH Datagram^ DO
96     BEGIN
97         Message_number := Datagram.Message_number + 1;
98         Timestring      := TIME_STRING ( Current_time_binary );
99     END;
100
101 WRITELN ( 'Subprocess about to send...', LF );
102
103 Send_message;
104
105 WAIT_ALL ( Jobs_port_value,
106            Green_light,
107            STATUS := Status_returned );
108     IF NOT ODD ( Status_returned ) THEN
109         WRITELN      ( 'WAIT status was: ',
110                        Status_returned :1);
111
112 WRITELN ( 'Subprocess about to receive...', LF );
113
114 Receive_message_and_display_it;
115
116 END;
117
118 { ----- }

```


EXAMPLES OF COMMUNICATION TECHNIQUES

```
119
120
121 PROGRAM Comm_2 (OUTPUT);
122
123 BEGIN
124
125 { get a pointer to the message packet }
126
127 CREATE_MESSAGE ( Message_ID,
128                 Datagram,
129                 STATUS := Status_returned );
130     IF NOT ODD ( Status_returned ) THEN
131     WRITELN      ( 'CREATE_MESSAGE status was: ',
132                 Status_returned :1);
133
134 Datagram^.Message_number := 0;
135
136 JOB_PORT ( Jobs_port_value,
137           STATUS := Status_returned );
138     IF NOT ODD ( Status_returned ) THEN
139     WRITELN      ( 'JOB_PORT status was: ',
140                 Status_returned :1);
141
142 { event for holding the subprocess }
143
144 CREATE_EVENT ( Green_light,
145              EVENT$CLEARED,
146              STATUS := Status_returned );
147     IF NOT ODD ( Status_returned ) THEN
148     WRITELN      ( 'CREATE_EVENT status was: ',
149                 Status_returned :1);
150
151 CREATE_PROCESS ( Sub_ID,
152                Sender,
153                STATUS := Status_returned );
154     IF NOT ODD ( Status_returned ) THEN
155     WRITELN      ( 'CREATE_PROCESS status was: ',
156                 Status_returned :1);
157
```

```

158 { hang around for the message to arrive }
159
160 WAIT_ANY ( Jobs_port_value,
161           STATUS := Status_returned );
162       IF NOT ODD ( Status_returned ) THEN
163           WRITELN      ( 'WAIT status was: ',
164                         Status_returned :1);
165
166 WRITELN ( 'Master process about to receive...', LF );
167
168 Receive_message_and_display_it;
169
170 { update the message packet }
171
172 WITH Datagram^ DO
173     BEGIN
174         Message_number := Datagram.Message_number + 1;
175         Timestring      := 'Closing message ...';
176     END;
177
178 WRITELN ( LF, 'Master process about to send...', LF );
179
180 Send_message;
181
182 { give the subprocess the green light to continue }
183
184 SIGNAL ( Green_light,
185         STATUS := Status_returned );
186       IF NOT ODD ( Status_returned ) THEN
187           WRITELN      ( 'SIGNAL status was: ',
188                         Status_returned :1);
189
190 { wait for the subprocess to die }
191
192 WAIT_ANY ( Sub_ID,
193           STATUS := Status_returned );
194       IF NOT ODD ( Status_returned ) THEN
195           WRITELN      ( 'WAIT status was: ',
196                         Status_returned :1);
197
198 END { of PROGRAM }.
199 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES.

E.2.1 Running COMM_2

Use the command procedure of the same name to build a system with COMM_2:

```
1 $      ! COMM_2.COM
2 $      ! Command procedure to build the VAXELN module COMM_2
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $      !
7 $      Compile:
8 $              @ELN_COMPILE_1 COMM_2
9 $      Link:
10 $              @ELN_LINK_1      COMM_2
11 $      System_build:
12 $              @ELN_EBUILD_1    COMM_2
13 $      Switch_off_verify:
14 $              SET NOVERIFY
15 $              EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole /nofile /noserver
2      program COMM_2 /debug
```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_2:

```
SH SYS
! Available: Pages: 17886, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:08.98 Total:  0 00:00:09.13
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_2, priority 16 is waiting.
!
CANCEL CONTROL
GO
! Job 4, process 1, program COMM_2 running.
!Subprocess about to send...
!
!Master process about to receive...
!
!Contents of the message received:
!Message number: 1
!Message text:  17-NOV-1858 00:00:10.86
!
!Master process about to send...
!
!Subprocess about to receive...
!
!Contents of the message received:
!Message number: 2
!Message text:  Closing message ...
! Job 4, process 1, program COMM_2 has exited.
!
EXIT
```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.3 EXAMPLE OF DATAGRAM COMMUNICATION - COMM_3.PAS

This program uses a datagram to communicate with another job. The datagram consists of a list of random numbers generated by a FUNCTION written in VAX FORTRAN.

Program COMM_3A runs first and calls INITIALIZATION_DONE to start COMM_3 rolling.

Datagrams are not the preferred mechanism for communication, circuits are much more reliable.

```
1  {
2  SOURCE:      COMM_3.PAS
3
4  PURPOSE:     Demonstrates communication between jobs using the
5                datagram method.
6                This program formats a message that includes an array
7                of random numbers generated by a VAX FORTRAN function
8
9  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_3
10
11 LINK:        $ LINK /DEBUG COMM_3, -
12              _$ ELN$:RTLSHARE /LIBRARY, -
13              _$ RTL /LIBRARY
14
15 BUILD:       $ EBUILD /NOEDIT COMM_3
16
17 NOTES:       1) Command procedure COMM_3.COM compiles, links and
18              builds this module into a system
19
20 }
21 MODULE Comm_3 [IDENT ('V1.000')];
22
23 { the external FORTRAN routine - note the attribute REFERENCE -
24   FORTRAN expects to receive the address of numeric data
25   in a call to one of its FUNCTIONS or SUBROUTINES
26 }
27
28 FUNCTION Random ( Seed : [REFERENCE] INTEGER ) : REAL; EXTERNAL;
29
30 CONST
31     Limit = 100;
32
33 { define the message packet }
34
35 TYPE
36     Message_packet = RECORD
37         Message_number : 0..MAXINT;
38         Binary_time     : LARGE INTEGER;
39         Some_numbers    : ARRAY[1..Limit] OF INTEGER;
40     END;
```

```

41
42  VAR
43      Other_port,
44      Jobs_port_value : PORT;
45      Telegram        : ^Message_packet;
46      Message_ID      : MESSAGE;
47      Name_ID         : NAME;
48      Status_returned : INTEGER;
49
50  { ----- }
51
52  PROGRAM Comm_3 (OUTPUT);
53
54  VAR
55      I                : INTEGER;
56      Randoms_seed     : INTEGER := 1987; { odd numbers are
57                                           preferred seed values }
58
59  BEGIN
60
61  { get a pointer to the message packet }
62
63  CREATE_MESSAGE ( Message_ID,
64                  Telegram,
65                  STATUS := Status_returned );
66
67      IF NOT ODD ( Status_returned ) THEN
68      WRITELN      ( 'CREATE_MESSAGE status was: ',
69                    Status_returned :1);
70
71  JOB_PORT ( Jobs_port_value,
72            STATUS := Status_returned );
73
74      IF NOT ODD ( Status_returned ) THEN
75      WRITELN      ( 'JOB_PORT status was: ',
76                    Status_returned :1);
77
78  { find the value of the other port - we only know its name so pass that
79    via the kernel to the name server. That will do the translation for
80    us.
81  }
82
83  TRANSLATE_NAME ( Other_port,
84                  'NEW YORK',
85                  NAME$UNIVERSAL,
86                  STATUS := Status_returned );
87
88      IF NOT ODD ( Status_returned ) THEN
89      WRITELN      ( 'TRANSLATE_NAME status was: ',
90                    Status_returned :1);
91

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

92 { load the message packet }
93
94 WITH Telegram^ DO
95   BEGIN
96     Message_number := 1;
97     GET_TIME ( Binary_time,
98               STATUS := Status_returned );
99     IF NOT ODD ( Status_returned ) THEN
100       WRITELN      ( 'WAIT status was: ',
101                     Status_returned :1);
102
103     FOR I := 1 TO Limit DO
104       Some_numbers[I] := TRUNC ( (Random (Randoms_seed) * 1000.0) );
105   END;
106
107 { dispatch the packet to 'NEW YORK' }
108
109 SEND ( Message_ID,
110        Other_port,
111        STATUS := Status_returned );
112
113     IF NOT ODD ( Status_returned ) THEN
114       WRITELN      ( 'SEND status was: ',
115                     Status_returned :1);
116
117 END { of PROGRAM }.
118 END { of MODULE };
```

EXAMPLES OF COMMUNICATION TECHNIQUES

The other job in this system runs the program COMM_3A.PAS

```

1  {
2  SOURCE:          COMM_3A.PAS
3
4  PURPOSE:         Demonstrates communication between jobs.
5                   This program runs before COMM_3 to establish a name
6                   for its job port then signals "INITIALIZATION DONE"
7
8  COMPILE:         $ EPASCAL /LIST /DEBUG COMM_3A
9
10 LINK:            $ LINK /DEBUG COMM_3A, -
11                  _$ ELN$:RTLSHARE /LIBRARY, -
12                  _$ RTL /LIBRARY
13
14 BUILD:           $ EBUILD /NOEDIT COMM_3
15
16 NOTES:           1) Command procedure COMM_3.COM compiles, links and
17                  builds this module into a system
18
19 }
20 MODULE Comm_3A [IDENT ('V1.000')];
21
22 CONST
23     Limit = 50;
24
25 TYPE
26     Message_packet = RECORD
27         Message_number : 0..MAXINT;
28         Binary_time     : LARGE_INTEGER;
29         Some_numbers    : ARRAY[1..Limit] OF INTEGER;
30     END;
31
32 VAR
33     Jobs_port_value : PORT;
34     Telegram         : ^Message_packet;
35     Message_ID       : MESSAGE;
36     Name_ID          : NAME;
37     Status_returned  : INTEGER;
38
39 { ----- }
40
41 PROGRAM Comm_3A (OUTPUT);
42
43 VAR
44     I                : INTEGER;
45

```


EXAMPLES OF COMMUNICATION TECHNIQUES

```

46 BEGIN
47
48 CREATE_MESSAGE ( Message_ID,
49                 Telegram,
50                 STATUS := Status_returned );
51
52     IF NOT ODD ( Status_returned ) THEN
53     WRITELN      ( 'CREATE_MESSAGE status was: ',
54                 Status_returned :1);
55
56 JOB_PORT ( Jobs_port_value,
57           STATUS := Status_returned );
58
59     IF NOT ODD ( Status_returned ) THEN
60     WRITELN      ( 'JOB_PORT status was: ',
61                 Status_returned :1);
62
63 { putting the port's name in the UNIVERSAL name table means the job
64   could run on a separate VAXELN node without modification. Datagrams
65   are not reliable especially over that sort of distance
66 }
67
68 CREATE_NAME ( Name_ID,
69              'NEW YORK',
70              Jobs_port_value,
71              TABLE := NAME$UNIVERSAL,
72              STATUS := Status_returned );
73
74     IF NOT ODD ( Status_returned ) THEN
75     WRITELN      ( 'CREATE_NAME status was: ',
76                 Status_returned :1);
77
78 { allow the other job running COMM_3 to start now }
79
80 INITIALIZATION_DONE ( STATUS := Status_returned );
81
82     IF NOT ODD ( Status_returned ) THEN
83     WRITELN      ( 'INITIALIZATION_DONE status was: ',
84                 Status_returned :1);
85

```

```

86 { wait for the message to arrive }
87
88 WAIT_ANY ( Jobs_port_value,
89           STATUS := Status_returned );
90
91     IF NOT ODD ( Status_returned ) THEN
92     WRITELN    ( 'WAIT status was: ',
93               Status_returned :1);
94
95 RECEIVE ( Message_ID,
96          Telegram,
97          Jobs_port_value,
98          STATUS := Status_returned );
99
100     IF NOT ODD ( Status_returned ) THEN
101     WRITELN    ( 'SEND status was: ',
102               Status_returned :1);
103
104 { display the message }
105
106 WITH Telegram^ DO
107 BEGIN
108     WRITELN ( 'Message number: ', Message_number:1 );
109     WRITELN ( 'Time of message: ', TIME_STRING ( Binary_time ) );
110     FOR I := 1 TO Limit DO
111     BEGIN
112         IF ( (I-1) MOD 10 = 0 ) THEN WRITELN;
113         WRITE ( Some_numbers[I]: 6 );
114     END;
115     WRITELN;
116 END;
117
118 END { of PROGRAM }.
119 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES

The FORTRAN function looks like this:

```
1  *
2  *      SOURCE:      RANDOM.FOR
3  *      USES:        VAX FORTRAN built-in RAN function
4  *
5      REAL FUNCTION Random ( Seed )
6
7      IMPLICIT NONE
8
9      INTEGER      Seed
10     REAL         Num
11
12     Random = RAN ( Seed )
13
14     RETURN
15     END
```

E.3.1 Running COMM_3

Use the command procedure of the same name to build a system with COMM_3:

```

1 $      ! COMM_3.COM
2 $      ! Command procedure to build the VAXELN module COMM_3
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $ Compile:
8 $      @ELN_COMPILE_1 COMM_3
9 $      @ELN_COMPILE_1 COMM_3A
10 $     SET VERIFY
11 $     FORTRAN -
12         /NOOPTIMIZE -
13         /DEBUG -
14         /LIST RANDOM
15 $ Link:
16 $     LINK -
17         /NOSYSLIB -
18         /DEBUG COMM_3, RANDOM -
19         ,ELN$:FRTLOBJECT /LIBRARY -
20         ,RTLSHARE /LIBRARY -
21         ,RTL /LIBRARY
22 $     SET NOVERIFY
23 $     @ELN_LINK_1 COMM_3A
24 $ System_build:
25 $     @ELN_EBUILD_1 COMM_3
26 $ Switch_off_verify:
27 $     SET NOVERIFY
28 $     EXIT

```

The .DAT file used by EBUILD looks like this:

```

1     characteristic /noconsole
2     program COMM_3 /debug
3     program COMM_3A /initialize /debug

```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_3:

```
SET TIME '02-MAR-1987 16:25:35.00'
GO
! Job 4, process 1, program COMM_3A running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]COMM_3.EXE;2
!
! Job 6, process 1, program COMM_3 needs attention.
! Module COMM_3
! 58:
!>>59: BEGIN
! 60:
! 61: { get a pointer to the message packet }
! 62:
! 63: CREATE_MESSAGE ( Message_ID,
GO
! Job 6, process 1, program COMM_3 running.
! Job 6, process 1, program COMM_3 has exited.
!
! Job 4, process 1, program COMM_3A running.
!Message number: 1
!Time of message: 2-MAR-1987 16:26:12.46
!
!      31      10      308      139      582      996      768      93      695      109
!    336     268     502     659     219     303     733     296     164     271
!    338     333      85     273     345     860     595     171     484     769
!    868     927     895     544       7     804     953     416     839     392
!       0      264     797     208     559     351     333     353     293     762
! Job 4, process 1, program COMM_3A has exited.
!
EXIT
```

E.4 EXAMPLE OF DATAGRAM COMMUNICATION - COMM_4.PAS

This program is a variation on COMM_3A and uses program arguments to get the port name. COMM_3 functions in a separate job as before

Program COMM_4 runs first and calls INITIALIZATION_DONE to start COMM_3 rolling.

Datagrams are not the preferred mechanism for communication, circuits are much more reliable.

```

1  {
2  SOURCE:          COMM_4.PAS
3
4  PURPOSE:         Demonstrates communication between jobs using
5                    datagrams.
6                    This program is a variation on COMM_3A and gets the
7                    port name from the program's arguments.
8
9  COMPILE:         $ EPASCAL /LIST /DEBUG COMM_4
10
11 LINK:            $ LINK /DEBUG COMM_4, -
12                  _$ ELN$:RTLSHARE /LIBRARY, -
13                  _$ RTL /LIBRARY
14
15 BUILD:           $ EBUILD /NOEDIT COMM_4
16
17 NOTES:           1) Command procedure COMM_4.COM compiles, links and
18                  builds this module into a system
19
20 }
21 MODULE Comm_4 [IDENT ('V1.000')];
22
23 CONST
24     Limit = 50;
25
26 TYPE
27     Message_packet = RECORD
28         Message_number : 0..MAXINT;
29         Binary_time     : LARGE_INTEGER;
30         Some_numbers    : ARRAY[1..Limit] OF INTEGER;
31     END;
32
33 VAR
34     Argument_number : INTEGER := 3;
35     Jobs_port_value : PORT;
36     Telegram        : ^Message_packet;
37     Message_ID      : MESSAGE;
38     Name_ID         : NAME;
39     Status_returned : INTEGER;
40     Port_name_arg   : VARYING_STRING(32);
41
42 { ----- }
```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

43
44 PROGRAM Comm_4 (OUTPUT);
45
46 VAR
47     I                : INTEGER;
48
49 BEGIN
50
51 CREATE_MESSAGE ( Message_ID,
52                 Telegram,
53                 STATUS := Status_returned );
54
55     IF NOT ODD ( Status_returned ) THEN
56     WRITELN      ( 'CREATE_MESSAGE status was: ',
57                   Status_returned :1);
58
59 JOB_PORT ( Jobs_port_value,
60           STATUS := Status_returned );
61
62     IF NOT ODD ( Status_returned ) THEN
63     WRITELN      ( 'JOB_PORT status was: ',
64                   Status_returned :1);
65
66 Port_name_arg := PROGRAM_ARGUMENT ( Argument_number );
67
68 CREATE_NAME ( Name_ID,
69              Port_name_arg, .
70              Jobs_port_value,
71              TABLE := NAME$UNIVERSAL,
72              STATUS := Status_returned );
73
74     IF NOT ODD ( Status_returned ) THEN
75     WRITELN      ( 'CREATE_NAME status was: ',
76                   Status_returned :1);
77
78 INITIALIZATION_DONE ( STATUS := Status_returned );
79
80     IF NOT ODD ( Status_returned ) THEN
81     WRITELN      ( 'INITIALIZATION_DONE status was: ',
82                   Status_returned :1);
83
84 WAIT_ANY ( Jobs_port_value,
85           STATUS := Status_returned );
86
87     IF NOT ODD ( Status_returned ) THEN
88     WRITELN      ( 'WAIT status was: ',
89                   Status_returned :1);

```

```

90
91 RECEIVE ( Message_ID,
92           Telegram,
93           Jobs_port_value,
94           STATUS := Status_returned );
95
96           IF NOT ODD ( Status_returned ) THEN
97             WRITELN      ( 'SEND status was: ',
98                           Status_returned :l);
99
100 WITH Telegram^ DO
101   BEGIN
102     WRITELN ( 'Message number: ', Message_number:l );
103     WRITELN ( 'Time of message: ', TIME_STRING ( Binary_time ) );
104     FOR I := 1 TO Limit DO
105       BEGIN
106         IF ( (I-1) MOD 10 = 0 ) THEN WRITELN;
107         WRITE ( Some_numbers[I]: 6 );
108       END;
109     WRITELN;
110   END;
111
112 END { of PROGRAM }.
113 END { of MODULE };

```


EXAMPLES OF COMMUNICATION TECHNIQUES

E.4.1 Running COMM_4

Use the command procedure of the same name to build a system with COMM_4:

```
1 $      ! COMM_4.COM
2 $      ! Command procedure to build the VAXELN module COMM_4
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $      !
7 $ Compile:
8 $          @ELN_COMPILE_1 COMM_3
9 $          @ELN_COMPILE_1 COMM_4
10 $      SET VERIFY
11 $      FORTRAN -
12          /NOOPTIMIZE -
13          /DEBUG -
14          /LIST RANDOM
15 $ Link:
16 $      LINK -
17          /NOSYSLIB -
18          /DEBUG COMM_3, RANDOM -
19          ,ELN$:FRTLOBJECT /LIBRARY -
20          ,RTLSHARE /LIBRARY -
21          ,RTL /LIBRARY
22 $      SET NOVERIFY
23 $          @ELN_LINK_1      COMM_4
24 $ System_build:
25 $          @ELN_EBUILD_1    COMM_4
26 $ Switch_off_verify:
27 $      SET NOVERIFY
28 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program COMM_3 /debug
program COMM_4 /initialize /debug /argument=("", "", "" "NEW YORK")
```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_4:

```
SH SYS
! Available: Pages: 17863, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:12.90 Total:  0 00:00:13.05
! Time used by past jobs: 0
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_4, priority 16 is waiting.
!
GO
! Job 4, process 1, program COMM_4 running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]COMM_3.EXE;3
!
! Job 6, process 1, program COMM_3 needs attention.
! Module COMM_3
! 58:
!>>59: BEGIN
! 60:
! 61: { get a pointer to the message packet }
! 62:
! 63: CREATE_MESSAGE ( Message_ID,
GO
! Job 6, process 1, program COMM_3 running.
! Job 6, process 1, program COMM_3 has exited.
!
! Job 4, process 1, program COMM_4 running.
!Message number: 1
!Time of message: 17-NOV-1858 00:01:01.33
!
!      31      10      308      139      582      996      768      93      695      109
!      336      268      502      659      219      303      733      296      164      271
!      338      333       85      273      345      860      595      171      484      769
!      868      927      895      544       7      804      953      416      839      392
!       0      264      797      208      559      351      333      353      293      762
! Job 4, process 1, program COMM_4 has exited.
!
EXIT
```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.5 EXAMPLE OF AREA DATA SHARING - COMM_5.PAS

This program creates a VAXELN AREA and uses a subprocess to manipulate data in the area - i.e. sharing the area with the master process.

```
1  {-----}
2  SOURCE:      COMM_5.PAS
3
4  PURPOSE:     To demonstrate creation of, and access to,
5                a VAXELN AREA
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_5, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG COMM_5, VAXELN-MODULES /LIBRARY, -
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT COMM_5
17
18 NOTES:       1) Command procedure COMM_5.COM compiles, links and
19                builds this module into a VAXELN system
20 {-----}
21 MODULE Create_area_01 [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report;
25
26 CONST
27     Master_process_greeting = 'This is from the master process';
28     Process_greeting        = 'This is from the created process';
29     Limit                    = 100;
30
31 TYPE
32     List_of_integers        = ARRAY [1..Limit] OF INTEGER;
33
34 VAR
35     Area_identity           : AREA;
36     Process_identity        : PROCESS;
37     Start_of_area          : ^List_of_integers;
38     Returned_status         : INTEGER;
39
40 {-----}
```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

41
42 PROCEDURE Write_contents_of_area;
43
44 VAR
45     I                : 1..Limit;
46
47 BEGIN
48     FOR I := 1 TO Limit DO
49         BEGIN
50             IF      ( (I-1) MOD 8 = 0)
51                 THEN WRITELN;
52             WRITE   ( Start_of_area^ [I] : 10 );
53             END;
54             WRITELN;
55     END;
56
57 {-----}
58
59 PROCEDURE Create_process_and_wait;
60
61 BEGIN
62     CREATE_PROCESS ( Process_identity,
63                     Area_accessor,
64                     STATUS := Returned_status );
65
66     Check_status_and_report ( Returned_status,
67                               'CREATE_PROCESS' );
68
69     WAIT_ANY ( Process_identity,
70              STATUS := Returned_status );
71
72     Check_status_and_report ( Returned_status,
73                               'WAIT_ANY' );
74     END;
75
76 {-----}

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```
77
78 PROCEDURE Issue_master_process_greeting;
79
80 BEGIN
81     WRITELN ( Master_process_greeting );
82 END;
83
84 {-----}
85
86 PROCEDURE Create_area_and_initialize;
87
88 VAR
89     I                : 1..Limit;
90
91 BEGIN
92
93     { create the area, get a pointer to it and give it a name }
94
95     CREATE_AREA (   Area_identity,
96                   Start_of_area,
97                   'Work_space',
98                   STATUS := Returned_status );
99
100     Check_status_and_report ( Returned_status,
101                              'CREATE_AREA' );
102
103     FOR I := 1 TO Limit DO
104         Start_of_area^ [I] := I;
105     END;
106
107 {----- END PROCEDURE DECLARATIONS ----- }
```

```

108
109 PROGRAM Create_area_01 (OUTPUT);
110
111 BEGIN
112     Issue_master_process_greeting;
113     Create_area_and_initialize;
114     Create_process_and_wait;
115
116     Issue_master_process_greeting;
117     Write_contents_of_area;
118 END.
119
120 {----- PROCESS BLOCK -----}
121
122 PROCESS_BLOCK Area_accessor;
123
124 VAR
125     I                : 1..Limit;
126
127 BEGIN
128     WRITELN ( Process_greeting );
129
130     Write_contents_of_area;
131
132     FOR I := 1 TO Limit DO
133         Start_of_area^ [I] := I * 5;
134
135     WRITELN ( 'Process completed write and modification to area' );
136
137 END { PROCESS };
138 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.5.1 Running COMM_5

Use the command procedure of the same name to build a system with COMM_5:

```
1 $      ! COMM_5.COM
2 $      ! Command procedure to build the VAXELN module COMM_5
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      SET VERIFY
8 $ Compile:
9 $      EPASCAL -
10         /LIST -
11         /DEBUG COMM_5, VAXELN-MODULES /LIBRARY
12 $ Link:
13 $      LINK -
14         /DEBUG COMM_5 -
15         ,VAXELN-MODULES /LIBRARY -
16         ,ELN$:RTLSHARE /LIBRARY -
17         ,ELN$:RTL /LIBRARY /INCLUDE= -
18         (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
19         OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
20 $      SET NOVERIFY
21 $ System_build:
22 $      @ELN_EBUILD_1 COMM_5
23
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program COMM_5 /debug
```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_5:

SH SYS

! Available: Pages: 17812, Page table slots: 51, Pool blocks: 273
 ! Time since SET TIME: Idle: 0 00:01:11.22 Total: 0 00:01:11.40
 ! Time used by past jobs: 0 00:00:00.02
 !
 ! Job 2, program XQDRIVER, priority 1 is waiting.
 ! Job 3, program EDEBUGREM, priority 3 is running.
 ! Job 4, program FALSERVER, priority 16 is waiting.
 ! Job 5, program COMM_5, priority 16 is waiting.
 !

CANCEL CONTROL

GO

! Job 5, process 1, program COMM_5 running.
 !This is from the master process
 !This is from the created process

!	1	2	3	4	5	6	7	8
!	9	10	11	12	13	14	15	16
!	17	18	19	20	21	22	23	24
!	25	26	27	28	29	30	31	32
!	33	34	35	36	37	38	39	40
!	41	42	43	44	45	46	47	48
!	49	50	51	52	53	54	55	56
!	57	58	59	60	61	62	63	64
!	65	66	67	68	69	70	71	72
!	73	74	75	76	77	78	79	80
!	81	82	83	84	85	86	87	88
!	89	90	91	92	93	94	95	96
!	97	98	99	100				

!Process completed write and modification to area

!This is from the master process

!	5	10	15	20	25	30	35	40
!	45	50	55	60	65	70	75	80
!	85	90	95	100	105	110	115	120
!	125	130	135	140	145	150	155	160
!	165	170	175	180	185	190	195	200
!	205	210	215	220	225	230	235	240
!	245	250	255	260	265	270	275	280
!	285	290	295	300	305	310	315	320
!	325	330	335	340	345	350	355	360
!	365	370	375	380	385	390	395	400
!	405	410	415	420	425	430	435	440
!	445	450	455	460	465	470	475	480
!	485	490	495	500				

! Job 5, process 1, program COMM_5 has exited.

!

EXIT

EXAMPLES OF COMMUNICATION TECHNIQUES

E.6 EXAMPLE OF DATA PASSING BY JOB ARGUMENT - COMM_6.PAS

This program creates a job - to run COMM_6A - and passes the time to that job through an argument in the call to CREATE_JOB

```
1  {
2  SOURCE:          COMM_6.PAS
3
4  PURPOSE:         Sets the system time then creates a job passing
5                   the current time to that job.
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG COMM_6
8
9  LINK:            $ LINK /DEBUG COMM_6, -
10                  _$ ELN$:RTLSHARE /LIBRARY, -
11                  _$ RTL /LIBRARY
12
13 BUILD:           $ EBUILD /NOEDIT COMM_6
14
15 NOTES:           1) Command procedure COMM_6.COM compiles, links and
16                  builds these programs into a system
17
18 }
19 MODULE Set_system_time [IDENT ('V1.000')];
20
21 VAR
22     From_users_input_time      : VARYING_STRING (23);
23     Status_from_call_to_SET_TIME : INTEGER;
24     The_current_binary_time_is  : LARGE_INTEGER;
25
26 { ----- }
27
28 PROCEDURE Prompt_user_for_time;
29
30 BEGIN
31     WRITELN ( 'Please enter time e.g. ',
32              '12-Dec-1986 15:47:30.00 > ' );
33     READLN  ( From_users_input_time );
34 END;
35
36 { ----- }
37
38 PROCEDURE And_get_the_binary_time;
39
40 BEGIN
41     The_current_binary_time_is :=
42         TIME_VALUE ( From_users_input_time );
43 END;
44
```

```

45 { ----- }
46
47 PROCEDURE Set_the_system_time;
48
49 BEGIN
50     SET_TIME      ( The_current_binary_time_is,
51                     STATUS := Status_from_call_to_SET_TIME );
52
53     IF NOT ODD     ( Status_from_call_to_SET_TIME ) THEN
54         WRITELN ( 'Call to SET_TIME failed with status: ',
55                     Status_from_call_to_SET_TIME );
56 END;
57
58 { ----- }
59
60 PROGRAM Set_system_time ( INPUT, OUTPUT );
61
62 VAR
63     Time_for_new_job : VARYING_STRING(23);
64     Status_returned  : INTEGER;
65     Jobs_port        : PORT;
66
67 BEGIN
68     Prompt_user_for_time;
69     And_get_the_binary_time;
70     Set_the_system_time;
71
72 { get time string for argument to CREATE_JOB }
73
74     And_get_the_binary_time;
75     Time_for_new_job := TIME_STRING ( The_current_binary_time_is );
76
77 CREATE_JOB ( Jobs_port,
78             'COMM_6A',
79             '', { null arguments for number 1 & 2 }
80             '',
81             Time_for_new_job,
82             STATUS := Status_returned );
83 IF NOT ODD ( Status_returned ) THEN
84     WRITELN ( 'Call to CREATE_JOB failed with status: ',
85                 Status_returned );
86
87 END { of PROGRAM }.
88 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES

The program COMM_6A reads its argument to collect the incoming data - a time string. It converts this time string to binary and adds 30 days to it before printing the new value. The code for COMM_6A looks like this:

```
1  {
2  SOURCE:      COMM_6A.PAS
3
4  PURPOSE:     Gets the time from program argument and displays it plus
5                30 days.
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_6A
8
9  LINK:        $ LINK /DEBUG COMM_6A, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT COMM_6
14
15 NOTES:       1) Command procedure COMM_6.COM compiles, links and
16                builds these programs into a system
17
18 }
19 MODULE Comm_6a [IDENT ('V1.000')];
20
21 CONST
22     Argument_required = 3;
23     Thirty_days       = '0030 00:00:00.00';
24
25 VAR
26     Argument_3,
27     Next_month_string : VARYING_STRING(23);
28     Interval_30_days,
29     Thirty_days_hence : LARGE_INTEGER;
30
31 { ----- }
32
33 PROGRAM Comm_6a ( OUTPUT );
34
35 BEGIN
36
37     Argument_3 := PROGRAM_ARGUMENT ( Argument_required );
38
39     Interval_30_days := TIME_VALUE ( Thirty_days );
```

```

40
41 { SUBTRACT interval because delta times are held internally as
42   NEGATIVE quadwords }
43 Thirty_days_hence := TIME_VALUE ( Argument_3 ) - Interval_30_days;
44
45 Next_month_string := TIME_STRING ( Thirty_days_hence );
46
47 WRITELN ( 'Program COMM_6A reporting..... ' );
48 WRITELN ( 'Time string passed to me was: ', Argument_3 );
49 WRITELN ( 'Adding 30 days to that gives: ', Next_month_string );
50
51 END { of PROGRAM }.
52 END { of MODULE };

```

E.6.1 Running COMM_6

Use the command procedure of the same name to build a system with COMM_6:

```

1 $      ! COMM_6.COM
2 $      ! Command procedure to build the VAXELN modules COMM_6
3 $      ! and COMM_6A
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $      @ELN_COMPILE_1 COMM_6
10 $      @ELN_COMPILE_1 COMM_6A
11 $ Link:
12 $      @ELN_LINK_1      COMM_6
13 $      @ELN_LINK_1      COMM_6A
14 $ System_build:
15 $      @ELN_EBUILD_1    COMM_6
16 $ Switch_off_verify:
17 $      SET NOVERIFY
18 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /nonetwork /nofile /noserver
2      program COMM_6 /debug
3      program COMM_6A /norun /debug

```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_6:

```
SH SYS
! Available: Pages: 17869, Page table slots: 51, Pool blocks: 268
! Time since SET TIME: Idle:  0 00:00:08.91 Total:  0 00:00:09.09
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_6, priority 16 is waiting.
!
GO
! Job 4, process 1, program COMM_6 running.
! Please enter time e.g. 12-Dec-1986 15:47:30.00 > 2-MAR-1987 17:14:20.00
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]COMM_6A.EXE;2
!
! Job 5, process 1, program COMM_6A needs attention.
! Module COMM_6A
! 34:
!>>35: BEGIN
! 36:
! 37: Argument_3 := PROGRAM_ARGUMENT ( Argument_required );
! 38:
! 39: Interval_30_days := TIME_VALUE ( Thirty_days );
! Job 4, process 1, program COMM_6 has exited.
!
GO
! Job 5, process 1, program COMM_6A running.
! Program COMM_6A reporting.....
! Time string passed to me was: 2-MAR-1987 17:14:20.00
! Adding 30 days to that gives: 1-APR-1987 17:14:20.00
! Job 5, process 1, program COMM_6A has exited.
!
EXIT
```

E.7 EXAMPLE OF USING CIRCUITS - COMM_7.PAS

This program accepts a circuit connection from COMM_7A which runs first. It waits for a message from COMM_7A

```

1  {-----}
2  SOURCE:      COMM_7.PAS
3
4  PURPOSE:     To demonstrate creation of a circuit between VAXELN jobs
5
6  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_7, -
7               _$ VAXELN-MODULES /LIBRARY
8
9  LINK:        $ LINK /DEBUG COMM_7, VAXELN-MODULES /LIBRARY, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ ELN$:RTL /LIBRARY /INCLUDE= -
12              _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
13              _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
14
15 BUILD:       $ EBUILD /NOEDIT COMM_7
16
17 NOTES:       1) Command procedure COMM_7.COM compiles, links
18              and builds this module into a VAXELN system
19  -----}
20 MODULE Comm_7 [IDENT ('V1.000')];
21
22 INCLUDE
23     Check_status_and_report;
24
25 TYPE
26     Message_type = ^VARYING_STRING (50);
27
28 {----- PROGRAM BLOCK -----}
29
30 PROGRAM Comm_7 (OUTPUT);
31
32 VAR
33     My_new_port,
34     New_jobs_port      : PORT;
35     Incoming_message   : Message_type;
36     Message_identity   : MESSAGE;
37     My_new_port_name   : NAME;
38     Returned_status    : INTEGER;
39

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

40 { MAIN program start }
41
42 BEGIN
43
44 { could have used the default job port instead of creating one }
45
46 CREATE_PORT (   My_new_port,
47                STATUS := Returned_status );
48
49                Check_status_and_report ( Returned_status,
50                                           'CREATE_PORT' );
51
52 { name the port and put the name in the UNIVERSAL table }
53
54 CREATE_NAME (   My_new_port_name,
55                'MASTER_PORT',
56                My_new_port,
57                TABLE := NAME$UNIVERSAL
58                STATUS := Returned_status );
59
60                Check_status_and_report ( Returned_status,
61                                           'CREATE_NAME' );
62
63 WRITELN ( 'First job about to accept circuit from second.....' );
64
65 { let the other job go }
66
67 INITIALIZATION_DONE ( STATUS := Returned_status );
68
69                Check_status_and_report ( Returned_status,
70                                           'INITIALIZATION_DONE' );
71
72 { process goes into wait state here for the connect circuit request
73   to arrive...}
74
75 ACCEPT_CIRCUIT ( My_new_port,
76                  STATUS := Returned_status );
77
78                Check_status_and_report ( Returned_status,
79                                           'ACCEPT_CIRCUIT' );

```

```
80
81 { wait for the message to arrive }
82
83 WAIT_ANY ( My_new_port,
84           STATUS := Returned_status );
85
86     Check_status_and_report ( Returned_status,
87                             'WAIT_ANY' );
88
89 WRITELN ( '.....back in the master process about to read message ' );
90
91 { collect the message and read it }
92
93 RECEIVE ( Message_identity,
94          Incoming_message,
95          My_new_port,
96          STATUS := Returned_status );
97
98     Check_status_and_report ( Returned_status,
99                             'RECEIVE' );
100
101 WRITELN ( 'Message: ', Incoming_message^ );
102
103 END { of PROGRAM }.
104 END { of MODULE };
```


EXAMPLES OF COMMUNICATION TECHNIQUES

This program, COMM_7A, throws out a circuit to COMM_7. With the connection established it sends a simple message by SENDING to its port. The code for COMM_7A looks like this:

```

1  {-----}
2  SOURCE:      COMM_7A.PAS
3
4  PURPOSE:     To demonstrate circuit connection with another job
5                This program throws out a circuit to COMM_7
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_7A, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG COMM_7A, VAXELN-MODULES /LIBRARY, -
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT COMM_7
17
18 NOTES:       1) Command procedure COMM_7.COM compiles, links
19                and builds this module into a VAXELN system
20 -----}
21 MODULE Comm_7a [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report;
25
26 TYPE
27     Message_type = ^VARYING_STRING (50);
28
29 {----- PROGRAM BLOCK -----}
30
31 PROGRAM Comm_7a (OUTPUT);
32
33 VAR
34     Returned_status      : INTEGER;
35     Closedown             : Message_type;
36     Message_identity     : MESSAGE;
37     New_port              : PORT;
38

```

```

39 { MAIN program start }
40
41 BEGIN
42
43 { could use the job's port instead }
44
45 CREATE_PORT      (   New_port,
46                     STATUS := Returned_status );
47
48     Check_status_and_report ( Returned_status,
49                               'CREATE_PORT' );
50
51 CONNECT_CIRCUIT (   New_port,
52                     DESTINATION_NAME := 'MASTER_PORT',
53                     STATUS := Returned_status );
54
55     Check_status_and_report ( Returned_status,
56                               'CONNECT_CIRCUIT' );
57
58 WRITELN ( 'This is from the new job' );
59
60 CREATE_MESSAGE (   Message_identity,
61                   Closedown,
62                   STATUS := Returned_status );
63
64     Check_status_and_report ( Returned_status,
65                               'CREATE_MESSAGE' );
66
67 Closedown^ := 'This is my closedown message';
68
69 SEND      (   Message_identity,
70              New_port,
71              STATUS := Returned_status );
72
73     Check_status_and_report ( Returned_status,
74                               'SEND' );
75 END { of PROGRAM }.
76 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.7.1 Running COMM_7

Use the command procedure of the same name to build a system with COMM_7:

```
1 $      ! COMM_7.COM
2 $      ! Command procedure to build the VAXELN modules
3 $      ! COMM_7 and COMM_7A into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG COMM_7, VAXELN-MODULES /LIBRARY
13 $ Compile2:
14 $      EPASCAL -
15          /LIST -
16          /DEBUG COMM_7A, VAXELN-MODULES /LIBRARY
17 $ Link1:
18 $      LINK -
19          /DEBUG COMM_7 -
20          ,VAXELN-MODULES /LIBRARY -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY /INCLUDE= -
23          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
24          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
25 $ Link2:
26 $      LINK -
27          /DEBUG COMM_7A -
28          ,VAXELN-MODULES /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY /INCLUDE= -
31          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
32          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
33 $ System_build:
34 $      EBUILD -
35          /NOEDIT COMM_7
36 $ Switch_off_verify:
37 $      SET NOVERIFY
38 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      CHARACTERISTIC /NOCONSOLE
2      PROGRAM COMM_7 /INITIALIZE /DEBUG
3      PROGRAM COMM_7A /DEBUG
```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_7:

```
SH SYS
! Available: Pages: 17749, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:10.36 Total:  0 00:00:10.52
! Time used by past jobs: 0
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_7, priority 16 is waiting.
!
GO
! Job 4, process 1, program COMM_7 running.
!First job about to accept circuit from second.....
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]COMM_7A.EXE;1
!
! Job 6, process 1, program COMM_7A needs attention.
!  Module  CREATE_CIRCUIT_A
!   40:
!>>41: BEGIN
!   42:
!   43: { could use the job's port instead }
!   44:
!   45: CREATE_PORT      (   New_port,
GO
! Job 6, process 1, program COMM_7A running.
! Job 6, process 1, program COMM_7A has exited.
!
! Job 4, process 1, program COMM_7 running.
!This is from the new job
!.....back in the master process about to read message
!Message: This is my closedown message
! Job 4, process 1, program COMM_7 has exited.
!
EXIT
```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.8 EXAMPLE OF USING CIRCUITS - COMM_8.PAS

This program accepts a circuit connection from COMM_8A which runs first. It sends messages to COMM_8A and receives messages from COMM_8A.

```
1  {-----}
2  SOURCE:      COMM_8.PAS
3
4  PURPOSE:     To demonstrate creation of a circuit between VAXELN
5                jobs. This program 'talks' to COMM_8A. Each sends
6                messages in a FOR loop, incrementing a message number
7                count each time a message is prepared.
8
9  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_8, -
10              _$ VAXELN-MODULES /LIBRARY
11
12  LINK:       $ LINK /DEBUG COMM_8, VAXELN-MODULES /LIBRARY, -
13              _$ ELN$:RTLSHARE /LIBRARY, -
14              _$ ELN$:RTL /LIBRARY /INCLUDE= -
15              _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
16              _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
17
18  BUILD:      $ EBUILD /NOEDIT COMM_8
19
20  NOTES:      1) Command procedure COMM_8.COM compiles, links
21              and builds this module into a VAXELN system
22  -----}
23  MODULE Comm_8 [IDENT ('V1.000')];
24
25  INCLUDE
26      Check_status_and_report;
27
28  CONST
29      LF      = '(10); { line-feed }
30      Limit = 10;
31
32  TYPE
33      Message_type = RECORD
34          Message_number : 0..MAXINT;
35          Data_array      : ARRAY[1..Limit] OF INTEGER;
36      END;
37
```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

38 {----- PROGRAM BLOCK -----}
39
40 PROGRAM Comm_8 (OUTPUT);
41
42 VAR
43     My_new_port      : PORT;
44     Outgoing_message;
45     Incoming_message : ^Message_type;
46     Message_identity : MESSAGE;
47     My_new_port_name : NAME;
48     I,
49     J,
50     Returned_status  : INTEGER;
51
52 { MAIN program start }
53
54 BEGIN
55
56 CREATE_PORT (    My_new_port,
57                 STATUS := Returned_status );
58
59     Check_status_and_report ( Returned_status,
60                             'CREATE_PORT' );
61
62 CREATE_NAME (    My_new_port_name,
63                 'MASTER_PORT',
64                 My_new_port,
65                 TABLE := NAMESUNIVERSAL,
66                 STATUS := Returned_status );
67
68     Check_status_and_report ( Returned_status,
69                             'CREATE_NAME' );
70
71 WRITELN ( 'This is from COMM_8...' );
72
73 INITIALIZATION_DONE ( STATUS := Returned_status );
74
75     Check_status_and_report ( Returned_status,
76                             'INITIALIZATION_DONE' );
77
78 { wait for COMM_8A's circuit request to arrive...}
79
80 ACCEPT_CIRCUIT ( My_new_port,
81                 STATUS := Returned_status );
82
83     Check_status_and_report ( Returned_status,
84                             'ACCEPT_CIRCUIT' );
85

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

86  FOR I := 1 TO 5 DO
87
88  BEGIN
89
90  CREATE_MESSAGE ( Message_identity,
91                    Outgoing_message,
92                    STATUS := Returned_status );
93
94                    Check_status_and_report ( Returned_status,
95                                                'CREATE_MESSAGE' );
96
97  WITH Outgoing_message^ DO
98
99  { fill array with data and set message number }
100
101  BEGIN
102    FOR J := 1 TO Limit DO Data_array[J] := J * I;
103    IF I = 1 THEN Message_number := 1
104    ELSE Message_number := Incoming_message^.Message_number + 1;
105  END;
106
107  SEND ( Message_identity,
108         My_new_port,
109         STATUS := Returned_status );
110
111         Check_status_and_report ( Returned_status,
112                                   'SEND' );
113
114  { wait for a reply...}
115
116  WAIT_ANY ( My_new_port,
117            STATUS := Returned_status );
118
119            Check_status_and_report ( Returned_status,
120                                      'WAIT_ANY' );
121
122  WRITELN ( 'Back in COMM_8 about to read message...' );
123
124  RECEIVE ( Message_identity,
125            Incoming_message,
126            My_new_port,
127            STATUS := Returned_status );
128
129            Check_status_and_report ( Returned_status,
130                                      'RECEIVE' );
131

```

```

132 WITH Incoming_message^ DO
133
134 { output reply }
135
136 BEGIN
137     WRITELN ( 'Message number: ', Message_number:1 );
138     FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
139     WRITELN ( LF );
140 END;
141
142 END; {FOR loop}
143
144 END { of PROGRAM }.
145 END { of MODULE };

```


EXAMPLES OF COMMUNICATION TECHNIQUES

This program, COMM_8A, throws out a circuit to COMM_8. It sends numbered messages to COMM_8.

```

1  {-----}
2  SOURCE:      COMM_8A.PAS
3
4  PURPOSE:     To demonstrate circuit connection with another job
5               This program 'talks' to COMM_8 and runs after COMM_8
6               calls INITIALIZATION_DONE.
7
8  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_8A, -
9               _$ VAXELN-MODULES /LIBRARY
10
11 LINK:        $ LINK /DEBUG COMM_8A, VAXELN-MODULES /LIBRARY, -
12               _$ ELN$:RTLSHARE /LIBRARY, -
13               _$ ELN$:RTL /LIBRARY /INCLUDE= -
14               _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
15               _$ OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
16
17 BUILD:       $ EBUILD /NOEDIT COMM_8
18
19 NOTES:       1) Command procedure COMM_8.COM compiles, links
20               and builds this module into a VAXELN system
21 -----}
22 MODULE Create_circuit_a [IDENT ('V1.000')];
23
24 INCLUDE
25     Check_status_and_report;
26
27 CONST
28     Padding = '                               ';
29     LF      = '(10);    { line-feed }
30     Limit   = 10;
31
32 TYPE
33     Message_type = RECORD
34         Message_number : 0..MAXINT;
35         Data_array      : ARRAY[1..Limit] OF INTEGER;
36     END;
37

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

38 {----- PROGRAM BLOCK -----}
39
40 PROGRAM Create_job_01 (OUTPUT);
41
42 VAR
43     New_port          : PORT;
44     Outgoing_message,
45     Incoming_message  : ^Message_type;
46     Message_identity  : MESSAGE;
47     My_new_port_name  : NAME;
48     I,
49     J,
50     Returned_status   : INTEGER;
51
52 { MAIN program start }
53
54 BEGIN
55
56 CREATE_PORT      (   New_port,
57                     STATUS := Returned_status );
58
59     Check_status_and_report ( Returned_status,
60                               'CREATE_PORT' );
61
62 CONNECT_CIRCUIT (   New_port,
63                     DESTINATION_NAME := 'MASTER_PORT',
64                     STATUS := Returned_status );
65
66     Check_status_and_report ( Returned_status,
67                               'CONNECT_CIRCUIT' );
68
69 FOR I := 1 TO 5 DO
70 BEGIN
71
72 WAIT_ANY (New_port,
73           STATUS := Returned_status );
74
75     Check_status_and_report ( Returned_status,
76                               'WAIT_ANY' );
77
78 RECEIVE ( Message_identity,
79           Incoming_message,
80           New_port,
81           STATUS := Returned_status );
82
83     Check_status_and_report ( Returned_status,
84                               'RECEIVE' );
85

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```
86  WRITELN ( Padding, 'This is from COMM_8A...' );
87
88  WITH Incoming_message^ DO
89  BEGIN
90    WRITELN ( Padding, 'Message number is: ', Message_number:1 );
91    WRITE   ( Padding );
92    FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
93    WRITELN ( LF );
94  END;
95
96  CREATE_MESSAGE (   Message_identity,
97                   Outgoing_message,
98                   STATUS := Returned_status );
99
100         Check_status_and_report ( Returned_status,
101                                   'CREATE_MESSAGE' );
102
103  WITH Outgoing_message^ DO
104
105  { multiply incoming data for reply and set new message number }
106
107  BEGIN
108    Message_number := Incoming_message^.Message_number + 1;
109    FOR J := 1 TO Limit DO Data_array[J] :=
110                          Incoming_message^.Data_array[J] * 2;
111  END;
112
113  SEND          (   Message_identity,
114                 New_port,
115                 STATUS := Returned_status );
116
117         Check_status_and_report ( Returned_status,
118                                   'SEND' );
119  END;
120
121  END { of PROGRAM }.
122  END { of MODULE };
```

E.8.1 Running COMM_8

Use the command procedure of the same name to build a system with COMM_8:

```

1  $      ! COMM_8.COM
2  $      ! Command procedure to build the VAXELN modules
3  $      ! COMM_8 and COMM_8A and build them into a system
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $ !
8  $      SET VERIFY
9  $ Compile1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG COMM_8, VAXELN-MODULES /LIBRARY
13 $ Compile2:
14 $      EPASCAL -
15          /LIST -
16          /DEBUG COMM_8A, VAXELN-MODULES /LIBRARY
17 $ Link1:
18 $      LINK -
19          /DEBUG COMM_8 -
20          ,VAXELN-MODULES /LIBRARY -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY /INCLUDE= -
23          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
24          OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
25 $ Link2:
26 $      LINK -
27          /DEBUG COMM_8A -
28          ,VAXELN-MODULES /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY /INCLUDE= -
31          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
32          OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
33 $      SET NOVERIFY
34 $ System_build:
35 $      @ELN_EBUILD_1 COMM_8
36 $ Switch_off_verify:
37 $      SET NOVERIFY
38 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1  CHARACTERISTIC /NOCONSOLE
2  PROGRAM COMM_8 /INITIALIZE /DEBUG
3  PROGRAM COMM_8A /DEBUG

```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_8:

```
SH SYS
! Available: Pages: 17749, Page table slots: 51, Pool blocks: 274
! Time since SET TIME: Idle:  0 00:00:52.83 Total:  0 00:00:52.98
! Time used by past jobs: 0
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_8, priority 16 is waiting.
!
GO
! Job 4, process 1, program COMM_8 running.
!This is from COMM_8...
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]COMM_8A.EXE;1
!
! Job 6, process 1, program COMM_8A needs attention.
! Module  CREATE_CIRCUIT_A
! 53:
!>>54: BEGIN
! 55:
! 56: CREATE_PORT      (   New_port,
! 57:                  STATUS := Returned_status );
! 58:
GO
! Job 6, process 1, program COMM_8A running.
!                               This is from COMM_8A...
!                               Message number is: 1
!                               1   2   3   4   5   6   7   8   9  10
!
!Back in COMM_8 about to read message...
!Message number: 2
!   2   4   6   8  10  12  14  16  18  20
!
!                               This is from COMM_8A...
!                               Message number is: 3
!                               2   4   6   8  10  12  14  16  18  20
!
!Back in COMM_8 about to read message...
!Message number: 4
!   4   8  12  16  20  24  28  32  36  40
!
!                               This is from COMM_8A...
!                               Message number is: 5
!                               3   6   9  12  15  18  21  24  27  30
```

EXAMPLES OF COMMUNICATION TECHNIQUES

```
!Back in COMM_8 about to read message...
!Message number: 6
!  6 12 18 24 30 36 42 48 54 60

!                                     This is from COMM_8A...
!                                     Message number is: 7
!                                     4  8 12 16 20 24 28 32 36 40

!Back in COMM_8 about to read message...
!Message number: 8
!  8 16 24 32 40 48 56 64 72 80

!                                     This is from COMM_8A...
!                                     Message number is: 9
!                                     5 10 15 20 25 30 35 40 45 50

! Job 6, process 1, program COMM_8A has exited.
!
! Job 4, process 1, program COMM_8 running.
!Back in COMM_8 about to read message...
!Message number: 10
! 10 20 30 40 50 60 70 80 90 100

! Job 4, process 1, program COMM_8 has exited.
!
EXIT
```

EXAMPLES OF COMMUNICATION TECHNIQUES

E.9 EXAMPLE OF USING CIRCUITS AND EXPEDITED MESSAGES - COMM_9.PAS

This program accepts a circuit connection from COMM_9A which runs first. It sends messages to COMM_9A until COMM_9A's port is full. Then an expedited message is sent. Good system design should not require the use of expedited messages. By default the sender process would enter a WAIT state if the receiver port is full.

```

1  {-----}
2  SOURCE:      COMM_9.PAS
3
4  PURPOSE:     To demonstrate creation of a circuit between VAXELN
5                jobs with an EXPEDITED message SEND
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_9, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG COMM_9, VAXELN-MODULES /LIBRARY, -
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT COMM_9
17
18 NOTES:       1) Command procedure COMM_9.COM compiles, links
19                and builds this module into a VAXELN system
20 -----}
21 MODULE Create_circuit [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report;
25
26 CONST
27     LF          = '(10);      { line-feed }
28     Limit       = 10;
29     Port_limit  = 8;
30
31 TYPE
32     Message_type = RECORD
33         Message_number : 0..MAXINT;
34         Data_array      : ARRAY[1..Limit] OF INTEGER;
35     END;
36
37     Expedited_type = RECORD
38         Expedite_string : VARYING_STRING(16);
39     END;
40

```

```

41 {----- PROGRAM BLOCK -----}
42
43 PROGRAM Create_job_01 (OUTPUT);
44
45 VAR
46     My_new_port      : PORT;
47     Expedited_message : ^Expedited_type;
48     Outgoing_message  : ^Message_type;
49     Message_identity  : MESSAGE;
50     My_new_port_name  : NAME;
51     I,
52     J,
53     Returned_status   : INTEGER;
54
55 { MAIN program start }
56
57 BEGIN
58
59     CREATE_PORT (   My_new_port,
60                   STATUS := Returned_status );
61
62     Check_status_and_report ( Returned_status,
63                               'CREATE_PORT' );
64
65     CREATE_NAME (   My_new_port_name,
66                   'MASTER_PORT',
67                   My_new_port,
68                   TABLE := NAME$UNIVERSAL,
69                   STATUS := Returned_status );
70
71     Check_status_and_report ( Returned_status,
72                               'CREATE_NAME' );
73
74     WRITELN ( 'This is from COMM_9...' );
75
76     INITIALIZATION_DONE ( STATUS := Returned_status );
77
78     Check_status_and_report ( Returned_status,
79                               'INITIALIZATION_DONE' );
80
81     ACCEPT_CIRCUIT ( My_new_port,
82                     STATUS := Returned_status );
83
84     Check_status_and_report ( Returned_status,
85                               'ACCEPT_CIRCUIT' );
86

```


EXAMPLES OF COMMUNICATION TECHNIQUES

```
87 { start pushing messages along the circuit }
88
89 FOR I := 1 TO Port_limit DO
90
91 BEGIN
92
93 CREATE_MESSAGE ( Message_identity,
94                 Outgoing_message,
95                 STATUS := Returned_status );
96
97         Check_status_and_report ( Returned_status,
98                                   'CREATE_MESSAGE' );
99
100
101 WITH Outgoing_message^ DO
102
103 BEGIN
104     FOR J := 1 TO Limit DO Data_array[J] := J * I;
105     Message_number := I;
106 END;
107
108 SEND ( Message_identity,
109        My_new_port,
110        STATUS := Returned_status );
111
112         Check_status_and_report ( Returned_status,
113                                   'SEND' );
114 END;
115
116 { having reached the limit for the port issue an EXPEDITED message }
117
118 CREATE_MESSAGE ( Message_identity,
119                 Expedited_message,
120                 STATUS := Returned_status );
121
122         Check_status_and_report ( Returned_status,
123                                   'CREATE_MESSAGE' );
124
```

```

125 WITH Expedited_message^ DO
126     Expedite_string := 'EXPEDITED !!!!!';
127
128 { the value 16 in the SIZE argument is required otherwise
129   the previous message size is assumed. A value > 16 will be
130   an error for a message typed EXPEDITED }
131
132 SEND ( Message_identity,
133        My_new_port,
134        SIZE      := 16,
135        EXPEDITE  := TRUE,
136        STATUS    := Returned_status );
137
138        Check_status_and_report ( Returned_status,
139                                'SEND' );
140
141 END { of PROGRAM }.
142 END { of MODULE };

```

EXAMPLES OF COMMUNICATION TECHNIQUES

This program, COMM_9A, throws out a circuit to COMM_9. Then it delays long enough for messages from COMM_9 to fill its port. Eventually it reads the messages testing the status returned by the procedure RECEIVE in case an abnormal message (e.g. expedited) is received.

```

1  {-----}
2  SOURCE:      COMM_9A.PAS
3
4  PURPOSE:     To demonstrate circuit connection with another job and
5                handling of an expedited message
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG COMM_9A, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG COMM_9A, VAXELN-MODULES /LIBRARY, -
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT COMM_9
17
18 NOTES:       1) Command procedure COMM_9.COM compiles, links
19                and builds this module into a VAXELN system
20 -----}
21 MODULE Create_circuit_a [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report;
25
26 CONST
27     Padding      = '                                ';
28     Port_limit   = 8;
29     LF           = '(10);  { line-feed }
30     Limit        = 10;
31     Interval     = '0000 00:00:10.00';
32
33 TYPE
34     Message_type = RECORD
35         Message_number : INTEGER;
36         Data_array      : ARRAY[1..Limit] OF INTEGER;
37     END;
38
39 VAR
40     KER$_EXPEDITED,
41     KER$_DISCONNECT,
42     KER$_SUCCESS      : [EXTERNAL, VALUE] INTEGER;
43

```

```

44 {----- PROGRAM BLOCK -----}
45
46 PROGRAM Create_job_01 (OUTPUT);
47
48 VAR
49     Binary_interval      : LARGE_INTEGER;
50     Expedited_message    : VARYING_STRING(44);
51                           { 44 is length of MESSAGE_TYPE }
52     New_port             : PORT;
53     Incoming_message     : ^Message_type;
54     Message_identity     : MESSAGE;
55     My_new_port_name     : NAME;
56     I,
57     J,
58     Returned_status      : INTEGER;
59
60 { MAIN program start }
61
62 BEGIN
63
64     CREATE_PORT      (   New_port,
65                        LIMIT  := Port_limit,
66                        STATUS := Returned_status );
67
68     Check_status_and_report ( Returned_status,
69                               'CREATE_PORT' );
70
71     CONNECT_CIRCUIT (   New_port,
72                        DESTINATION_NAME := 'MASTER_PORT',
73                        STATUS := Returned_status );
74
75     Check_status_and_report ( Returned_status,
76                               'CONNECT_CIRCUIT' );
77
78 { place process in wait state so that PORT will become filled
79   with unread messages from COMM_9 }
80
81     Binary_interval := TIME_VALUE ( Interval );
82
83     WAIT_ANY (TIME := Binary_interval,
84              STATUS := Returned_status );
85
86     Check_status_and_report ( Returned_status,
87                               'WAIT_ANY' );
88

```

EXAMPLES OF COMMUNICATION TECHNIQUES

```

89  FOR I := 1 TO 10 DO
90  BEGIN
91
92  WRITELN ( Padding, 'This is from COMM_9A...' );
93
94  RECEIVE ( Message_identity,
95            Incoming_message,
96            New_port,
97            STATUS := Returned_status );
98
99  { Usual status checks are replaced by specific checks...
100    Can't use the KER$ codes with CASE statement - won't compile with
101    this TYPE for a CASE because they are not of the same type as the
102    CASE expression }
103
104  IF Returned_status = KER$_EXPEDITED THEN
105  BEGIN
106      WRITELN ( Padding, '***** EXPEDITED MESSAGE RECEIVED *****' );
107
108      { typecast to 42 bytes because varying_string has 16 bit header too -
109        the message type has a total of 44 bytes allocated to it.
110        Must be typecast because the data type of the message coming in
111        is incompatible with the buffer in this program }
112
113      Expedited_message := Incoming_message^::VARYING_STRING(42);
114      WRITELN ( Padding, 'Expedited message is: ' );
115      WRITELN ( Padding, Expedited_message, LF );
116  END;
117
118  IF Returned_status = KER$_SUCCESS THEN
119
120      WITH Incoming_message^ DO
121      BEGIN
122          WRITELN ( Padding, 'Message number is: ', Message_number:1 );
123          WRITE   ( Padding );
124          FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
125          WRITELN ( LF );
126      END;
127
128  IF Returned_status = KER$_DISCONNECT THEN
129      WRITELN ( Padding, 'Job COMM_9 has disconnected the circuit...' );
130
131  END;
132
133  END { of PROGRAM }.
134  END { of MODULE };

```

E.9.1 Running COMM_9

Use the command procedure of the same name to build a system with COMM_9:

```

1 $      ! COMM_9.COM
2 $      ! Command procedure to build the VAXELN modules
3 $      ! COMM_9 and COMM_9A and build them into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG COMM_9, VAXELN-MODULES /LIBRARY
13 $ Compile2:
14 $      EPASCAL -
15          /LIST -
16          /DEBUG COMM_9A, VAXELN-MODULES /LIBRARY
17 $ Link1:
18 $      LINK -
19          /DEBUG COMM_9 -
20          ,VAXELN-MODULES /LIBRARY -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY /INCLUDE= -
23          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
24          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
25 $ Link2:
26 $      LINK -
27          /DEBUG COMM_9A -
28          ,VAXELN-MODULES /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY /INCLUDE= -
31          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
32          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
33 $      SET NOVERIFY
34 $ System_build:
35 $      @ELN_EBUILD_1 COMM_9
36 $ Switch_off_verify:
37 $      SET NOVERIFY
38 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1 CHARACTERISTIC /NOCONSOLE
2 PROGRAM COMM_9 /INITIALIZE /DEBUG
3 PROGRAM COMM_9A /DEBUG

```

EXAMPLES OF COMMUNICATION TECHNIQUES

Output from COMM_9

SH SYS

! Available: Pages: 17750, Page table slots: 51, Pool blocks: 274
! Time since SET TIME: Idle: 0 00:01:09.06 Total: 0 00:01:09.23
! Time used by past jobs: 0

!

! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program COMM_9, priority 16 is waiting.

!

GO

! Job 4, process 1, program COMM_9 running.
! This is from COMM_9...
! Loading traceback data from: DISK\$INSTRUCT:[SHONE.VAXELN]COMM_9A.EXE;1

!

! Job 6, process 1, program COMM_9A needs attention.

! Module CREATE_CIRCUIT_A

! 61:

!>>62: BEGIN

! 63:

! 64: CREATE_PORT (New_port,

! 65: LIMIT := Port_limit,

! 66: STATUS := Returned_status);

GO

! Job 6, process 1, program COMM_9A running.
! Job 4, process 1, program COMM_9 has exited.

!

! Job 6, process 1, program COMM_9A running.

! This is from COMM_9A...

! ***** EXPEDITED MESSAGE RECEIVED *****

! Expedited message is:

! EXPEDITED !!!!!!!

!

This is from COMM_9A...

! Message number is: 1

! 1 2 3 4 5 6 7 8 9 10

!

This is from COMM_9A...

! Message number is: 2

! 2 4 6 8 10 12 14 16 18 20

!

This is from COMM_9A...

! Message number is: 3

! 3 6 9 12 15 18 21 24 27 30

EXAMPLES OF COMMUNICATION TECHNIQUES

```
!           This is from COMM_9A...
!           Message number is: 4
!           4   8  12  16  20  24  28  32  36  40

!           This is from COMM_9A...
!           Message number is: 5
!           5  10  15  20  25  30  35  40  45  50

!           This is from COMM_9A...
!           Message number is: 6
!           6  12  18  24  30  36  42  48  54  60

!           This is from COMM_9A...
!           Message number is: 7
!           7  14  21  28  35  42  49  56  63  70

!           This is from COMM_9A...
!           Message number is: 8
!           8  16  24  32  40  48  56  64  72  80

!           This is from COMM_9A...
!           Job COMM_9 has disconnected the circuit...
! Job 6, process 1, program COMM_9A has exited.
!
EXIT
```


APPENDIX F
NETWORK EXAMPLE

F.1 EXAMPLE OF NETWORK USE - NET_1.PAS

This program shows the dynamic downline loading of an executable image. The image to be loaded is compiled and linked in the usual way but not built into the system. When EBUILD is invoked the guaranteed image list must include the PRGLOADER image.

```
1  {-----
2  SOURCE:      NET_1.PAS
3
4  PURPOSE:     This program loads, dynamically, TODAY.EXE from the
5                host VAX and runs the program in a new job created
6                by this program for it
7                TODAY.EXE, written in VAX FORTRAN returns:
8                a) day
9                b) month
10               c) year
11               d) week
12               e) year day
13               f) century day
14               for the current date
15
16  COMPILE:     $ EPASCAL /LIST /DEBUG NET_1, VAXELN-MODULES /LIB -
17               _$ ,ELN$:RTLOBJECT /LIBRARY
18
19               $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY
20
```

NETWORK EXAMPLE

```

21 LINK:          $ LINK /DEBUG NET_1, -
22                _$ ELN$:RTLSHARE /LIBRARY, -
23                _$ ELN$:RTL /LIBRARY /INCLUDE= -
24                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
25                _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
26
27                $ LINK /DEBUG /NOSYSLIB TODAY, -
28                _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
29                _$ RTL /LIBRARY
30
31 BUILD:          $ EBUILD /NOEDIT NET_1
32
33 NOTES:          1) Command procedure NET_1.COM compiles, links and
34                  builds these programs into a system
35
36 }
37 MODULE Set_system_time [IDENT ('V1.000')];
38
39 INCLUDE
40     Check_status_and_report,
41     $LOADER_UTILITY;
42
43 CONST
44     Wait_period      = '0000 00:00:05.00';
45
46 VAR
47     Status_returned  : INTEGER;
48     Jobs_port        : PORT;
49     Program_name     : VARYING_STRING(40) := 'TODAY';
50
51     From_users_input_time      : VARYING_STRING (23);
52     Status_from_call_to_SET_TIME : INTEGER;
53     Binary_interval,
54     The_current_binary_time_is  : LARGE_INTEGER;
55     Loaded_jobs_port           : PORT;
56 { ----- }

```

```

57
58 PROCEDURE Prompt_user_for_time;
59
60 BEGIN
61     WRITELN ( 'Please enter time e.g. ',
62              '12-Dec-1986 15:47:30.00 > ' );
63     READLN ( From_users_input_time );
64 END;
65
66 { ----- }
67
68 PROCEDURE And_get_the_binary_time;
69
70 BEGIN
71     The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
72 END;
73
74 { ----- }
75
76 PROCEDURE Set_the_system_time;
77
78 BEGIN
79     SET_TIME      ( The_current_binary_time_is,
80                    STATUS := Status_from_call_to_SET_TIME );
81
82     IF NOT ODD    ( Status_from_call_to_SET_TIME ) THEN
83         WRITELN ( 'Call to SET_TIME failed with status: ',
84                  Status_from_call_to_SET_TIME );
85 END;
86
87 { ----- }
88
89 PROGRAM Set_system_time ( INPUT, OUTPUT );
90
91 BEGIN
92     Prompt_user_for_time;
93     And_get_the_binary_time;
94     Set_the_system_time;

```

NETWORK EXAMPLE

```

95
96 ELN$LOAD_PROGRAM ( '1.241::DISK$COURSEDSK:[SHONE]TODAY.EXE',
97                     Program_name,
98                     FALSE,                {Kernel_mode}
99                     TRUE,                 {Start_with_debug}
100                    FALSE,                {Power_recovery}
101                    2,                    {Kernel_stack_size}
102                    1,                    {Initial_user_stack_size}
103                    0,                    {Message_limit}
104                    16,                   {Job_priority}
105                    8,                    {Process_priority}
106                    STATUS := Status_returned );
107
108     Check_status_and_report ( Status_returned,
109                              'ELN$LOAD_PROGRAM' );
110
111 CREATE_JOB ( Loaded_jobs_port,
112             Program_name,
113             STATUS := Status_returned );
114
115     Check_status_and_report ( Status_returned,
116                              'CREATE_JOB' );
117
118 Binary_interval := TIME_VALUE ( Wait_period );
119
120 WAIT_ANY ( TIME := Binary_interval,
121           STATUS := Status_returned );
122
123     Check_status_and_report ( Status_returned,
124                              'WAIT_ANY' );
125
126 END { of PROGRAM }.
127 END { of MODULE };

```

F.1.1.1 Running NET_1

Use the command procedure of the same name to build a system with NET_1:

```

1 $      ! NET_1.COM
2 $      ! Command procedure to build the EPASCAL module NET_1
3 $      ! and the VAX FORTRAN program TODAY into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile_NET_1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG NET_1, VAXELN-MODULES /LIBRARY -
13          ,ELN$:RTLOBJECT /LIBRARY
14 $ Compile_Today:
15 $      FORTRAN -
16          /LIST -
17          /NOOPTIMIZE -
18          /DEBUG TODAY
19 $ Link_NET_1:
20 $      LINK -
21          /DEBUG NET_1 -
22          ,VAXELN-MODULES /LIBRARY -
23          ,ELN$:RTLSHARE /LIBRARY -
24          ,ELN$:RTL /LIBRARY /INCLUDE= -
25          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
26          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
27 $ Link_Today:
28 $      LINK -
29          /DEBUG -
30          /NOSYSLIB TODAY -
31          ,ELN$:FRTLOBJECT /LIBRARY -
32          ,ELN$:RTLSHARE /LIBRARY -
33          ,ELN$:RTL /LIBRARY
34 $ System_build:
35 $      EBUILD -
36          /NOEDIT NET_1
37 $ Switch_off_verify:
38 $      SET NOVERIFY
39 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /image_list=(PRGLOADER)
2      program NET_1 /debug

```

NETWORK EXAMPLE

Output from NET_1:

SH SYS

! Available: Pages: 1359, Page table slots: 49, Pool blocks: 256
! Time since SET TIME: Idle: 0 00:01:00.93 Total: 0 00:01:01.23
! Time used by past jobs: 0 00:00:00.02

!

! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSESERVER, priority 16 is waiting.
! Job 5, program NET_1, priority 16 is waiting.

!

GO

! Job 5, process 1, program NET_1 running.
! Please enter time e.g. 12-Dec-1986 15:47:30.00 > 16-MAR-1987 14:08:00.00

! Job 5, process 1, program NET_1 running.
! Loading traceback data from: 1.241::DISK\$COURSEDSK:[SHONE]TODAY.EXE;1

!

! Job 6, process 1, program TODAY needs attention.
! Module TODAY\$MAIN

! 18:

!>>19: IMPLICIT NONE

! 20:

! 21: INTEGER Month_length(12) /31, 28, 31, 30, 31, 30,

! 22: 1 31, 31, 30, 31, 30, 31/

! 23:

! Job 5, process 1, program NET_1 has exited.

!

GO

! Job 6, process 1, program TODAY running.

!

!

! 14:08:24 Monday 16th March 1987 W 11, Y 75, C 31486

!

!

! Job 6, process 1, program TODAY has exited.

!

EXIT

APPENDIX G

FILE HANDLING

G.1 EXAMPLE OF FILE ACCESS ROUTINES - FILE_1.PAS

This first example was written by the Real-time Product Support Unit of the Atlanta Customer Support Center to whom the author is grateful.

```

{                                                                 }
{ Program demonstrating various file access functions           }
{                                                                 }
MODULE the_file_access_tests;
{                                                                 }
{ Include utilities used by this program                       }
{                                                                 }
INCLUDE $DISK_UTILITY, $FILE_UTILITY, $ELNMSG, $PASCALMSG, $KERNELMSG,
      $OTSMSG, $GET_MESSAGE_TEXT;

VAR  working_file          : TEXT;
     test_text, check_text : VARYING_STRING (80);
     pointer_to_directory  : ^ELN$DIR_FILE;
     bad_block_list        : DSK$_BADLIST(10);
     volume_name, directory_name, file_name,
     name_of_file_to_delete : VARYING_STRING (255);
     result                : INTEGER;
     result_string         : VARYING_STRING(255);
     which_flag            : GET_STATUS_FLAGS;

PROGRAM file_access_test(OUTPUT);

VAR i : INTEGER;
```


FILE HANDLING

```
BEGIN
{
{ Use the empty set so that you'll get all the information.
{
    which_flag := [];
{
{ First, initialize the volume. The volume will be referenced as DISK$SCRATCH
{
    INIT_VOLUME('DUAL',
                'SCRATCH',
                VERIFIED := FALSE,
                BAD_LIST := bad_block_list::DSK$_BADLIST(0),
                STATUS := result);
    ELN$GET_STATUS_TEXT(result, which_flag, result_string);
    WRITELN('The status of initializing the volume was');
    WRITELN(result_string);
{
{ Mount the volume and get the status
{
    MOUNT_VOLUME('DUAL',,result);
    ELN$GET_STATUS_TEXT(result, which_flag, result_string);
    WRITELN('The status of mounting the volume was');
    WRITELN(result_string);
{
{ Then, create a directory, get the status and create a variable to get the
{ information about the files in a directory.
{
    CREATE_DIRECTORY('DISK$SCRATCH:[RDB]', result);
    ELN$GET_STATUS_TEXT(result, which_flag, result_string);
    WRITELN('The status of creating the directory was');
    WRITELN(result_string);
    NEW(pointer_to_directory);
```

```

{                                                                                               }
{ Now, create ten new files and put a line of text into each                                   }
{                                                                                               }
{                                                                                               }
  FOR i := 1 TO 10 DO
    BEGIN
      WRITELN('This is loop number ', i:2);
      OPEN(working_file,
        FILE_NAME      := 'DISK$SCRATCH:[RDB]TEST.TXT',
        HISTORY        := HISTORY$NEW,
        ACCESS_METHOD  := ACCESS$SEQUENTIAL,
        SHARING        := SHARE$READWRITE);
      REWRITE(working_file);
      test_text := 'This is a test of the writing a file';
      WRITELN(working_file, test_text);
      CLOSE(working_file);
    END;
{                                                                                               }
{ Open the last one and read from it                                                         }
{                                                                                               }
{                                                                                               }
  OPEN(working_file,
    FILE_NAME      := 'DISK$SCRATCH:[RDB]TEST.TXT',
    HISTORY        := HISTORY$OLD,
    ACCESS_METHOD  := ACCESS$SEQUENTIAL,
    SHARING        := SHARE$READONLY);
  RESET(working_file);
  READLN(working_file, check_text);
  WRITELN('The text read was "', check_text, '"');
  CLOSE(working_file);
{                                                                                               }
{ Then, get a listing of the directory                                                         }
{                                                                                               }
{                                                                                               }
  DIRECTORY_OPEN(pointer_to_directory,
    'DISK$SCRATCH:[RDB]TEST.TXT;* ',
    volume_name,
    directory_name,
    result);
  DIRECTORY_LIST(pointer_to_directory, volume_name, file_name, result);

```

FILE HANDLING

```
{                                                                                               }
{ Use the directory to delete the ten files                                                    }
{                                                                                               }
  WHILE (ODD(result)) DO
    BEGIN
      name_of_file_to_delete := volume_name + directory_name + file_name;
      WRITELN('Delete ---> ', name_of_file_to_delete);
      DELETE_FILE(name_of_file_to_delete, result);
      ELN$GET_STATUS_TEXT(result, which_flag, result_string);
      WRITELN('The status of deleting the file was');
      WRITELN(result_string);
      DIRECTORY_LIST(pointer_to_directory, volume_name, file_name, result);
    END;
  WRITELN;
  WRITELN('All files deleted !!!');
END;
END.
```

G.1.1.1 Running FILE_1

Use the command procedure of the same name to build a system with FILE_1

```

1 $      ! FILE_1.COM
2 $      ! Command procedure to build the VAXELN module FILE_1
3 $      ! into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG FILE_1, ELN$:RTLOBJECT /LIBRARY
13 $ Link:
14 $      LINK -
15          /DEBUG FILE_1 -
16          ,ELN$:RTLSHARE /LIBRARY -
17          ,ELN$:RTL /LIBRARY /INCLUDE= -
18          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
19          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
20 $ System_build:
21 $      EBUILD -
22          /NOEDIT FILE_1
23 $ Switch_off_verify:
24 $      SET NOVERIFY
25 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1 characteristic /noconsole
2 program FILE_1 /debug
3 device DUA /register=%0772150 /vector=%0154

```

FILE HANDLING

G.2 EXAMPLE OF REMOTE FILE HANDLING - FILE_2..FOR

This VAX FORTRAN program reads an ASCII text file from a remote VAX node and displays the contents of the file on the user's terminal:

```
1  *-----
2  * SOURCE:                FILE_2.FOR
3  *
4  * PURPOSE:               This VAX FORTRAN program shows remote file access
5  *                       from VAXELN. It reads a file called ELEMENTS.DAT
6  *                       whose contents look like:
7  *
8  *                       Ac 89227.00000Actinium
9  *                       Al 13 26.98150Aluminium
10 *                       Am 95243.00000Americium
11 *                       . . . . .
12 *                       . . . . .
13 *                       Yb 70173.04000Ytterbium
14 *                       Y  39 88.90500Yttrium
15 *                       Zn 30 65.37000Zinc
16 *                       Zr 40 91.22000Zirconium
17 *
18 *
19 * COMPILE:               $ FORTRAN /LIST /DEBUG /NOOPTIMIZE FILE_2
20 *
21 * LINK:                  $ LINK /DEBUG FILE_2, -
22 *                        _$ ELN$:FRTLOBJECT /LIBRARY, -
23 *                        _$ ELN$:RTLSHARE /LIBRARY, -
24 *                        _$ ELN$:RTL /LIBRARY
25 *
26 * BUILD:                 $ EBUILD /NOEDIT FILE_2
27 *
28 * NOTES:                 1) Command procedure FILE_2.COM compiles, links
29 *                       and builds this module into a VAXELN system
30 * -----
```

```

31
32      PROGRAM VAXELN_file_IO
33
34      IMPLICIT NONE
35
36      INTEGER      First_element,
37      1            Last_element,
38      1            Unit_number,
39      1            Terminal
40
41      * When 'they' invent more elements change this parameter value
42
43      PARAMETER    ( First_element = 1,
44      1            Last_element  = 103,
45      1            Unit_number   = 4,
46      1            Terminal      = 6 )
47
48      * change this file spec if not on node 1.241 and device and
49      * directory details are different
50
51      CHARACTER *50 File_spec
52      1            /'1.241::DISK$COURSEDSK:[SHONE]ELEMENTS.DAT'/
53
54      CHARACTER      Symbol *2,
55      1            Element_name *12
56
57      INTEGER      Element_number,
58      1            Atomic_number
59
60      REAL          Atomic_weight
61
62      OPEN    ( Unit_number,
63      1        FILE      = File_spec,
64      1        STATUS    = 'OLD' )
65
66      * Read file and display to terminal
67
68      DO Element_number = First_element, Last_element, 1
69
70          READ ( Unit_number,
71      1          '(A, I3, F9.5, A)',
72      1          END = 999 )
73      1          Symbol, Atomic_number, Atomic_weight, Element_name
74
75      * subroutine call to convert lowercase to uppercase
76      * in 'Element_name'
77
78      CALL LC2UC ( Element_name,
79      1          Element_name )
80

```

FILE HANDLING

```

81      WRITE ( Terminal,
82      1      '(1X, ''Details of element: '', A /
83      1      1X, ''Symbol: '', A,
84      1      '', atomic number: '', I3,
85      1      '', atomic weight: '', F9.5 / )')
86      1      Element_name, Symbol, Atomic_number, Atomic_weight
87
88      END DO
89
90  999    CLOSE ( Unit_number )
91      END
92
93      SUBROUTINE LC2UC (INSTR,OUTSTR)
94  *
95  *      Subroutine to convert lowercase letters to uppercase
96  *
97  *      Input required is string and output location
98  *
99      IMPLICIT NONE
100
101      INTEGER      K,
102      1      Key
103      CHARACTER    INSTR*(*),
104      1      OUTSTR*(*),
105      1      Lcase *26,
106      1      Ucase *26
107
108      DATA Lcase /'abcdefghijklmnopqrstuvwxyz'/
109      DATA Ucase /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
110
111      DO K = 1, LEN(INSTR), 1
112
113          Key = INDEX (Lcase, Instr(K:K))
114
115          IF (Key .GT. 0) THEN
116              Outstr(K:K) = Ucase(Key:Key)
117          ELSE
118              Outstr(K:K) = Instr(K:K)
119          END IF
120
121      END DO
122
123      RETURN
124      END

```

G.2.1 Running FILE_2

Use the command procedure of the same name to build a system with FILE_2

```

1 $      ! FILE_2.COM
2 $      ! Command procedure to build the VAXELN module FILE_2
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      SET VERIFY
8 $ Compile:
9 $      FORTRAN -
10         /LIST -
11         /NOOPTIMIZE -
12         /DEBUG FILE_2
13 $ Link:
14 $      LINK -
15         /DEBUG FILE_2 -
16         ,ELN$:FRTLOBJECT /LIBRARY -
17         ,ELN$:RTLSHARE /LIBRARY -
18         ,ELN$:RTL /LIBRARY
19 $ System_build:
20 $      EBUILD -
21         /NOEDIT FILE_2
22 $ Switch_off_verify:
23 $      SET NOVERIFY
24 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole
2      program FILE_2 /debug

```


FILE HANDLING

Output from FILE_2:

SH SYS

! Available: Pages: 17824, Page table slots: 51, Pool blocks: 274
! Time since SET TIME: Idle: 0 00:00:49.64 Total: 0 00:00:49.82
! Time used by past jobs: 0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program FILE_2, priority 16 is waiting.
!

GO

! Job 5, process 1, program FILE_2 running.
!Details of element: ACTINIUM
!Symbol: Ac, atomic number: 89, atomic weight: 227.00000
!
!Details of element: ALUMINIUM
!Symbol: Al, atomic number: 13, atomic weight: 26.98150
!
!Details of element: AMERICIUM
!Symbol: Am, atomic number: 95, atomic weight: 243.00000
!
! <output suppressed to save space>
!
!Details of element: YTTRIUM
!Symbol: Y, atomic number: 39, atomic weight: 88.90500
!
!Details of element: ZINC
!Symbol: Zn, atomic number: 30, atomic weight: 65.37000
!
!Details of element: ZIRCONIUM
!Symbol: Zr, atomic number: 40, atomic weight: 91.22000
!
! Job 5, process 1, program FILE_2 has exited.
!

EXIT

APPENDIX H

DEVICE DRIVER EXAMPLES

H.1 EXAMPLE OF SUPPLIED DRIVER - ELN\$:LPVDRIVER.PAS

This is a listing of ELN\$:LPVDRIVER.PAS supplied with VAXELN.

```
module lpvdriver [ident('V2.0-00')];
```

```
{*****
{*
{* Copyright (c) 1984, 1985
{* by DIGITAL Equipment Corporation, Maynard, Mass.
{*
{* This software is furnished under a license and may be used and copied
{* only in accordance with the terms of such license and with the
{* inclusion of the above copyright notice. This software or any other
{* copies thereof may not be provided or otherwise made available to any
{* other person. No title to and ownership of the software is hereby
{* transferred.
{*
{* The information in this software is subject to change without notice
{* and should not be construed as a commitment by DIGITAL Equipment
{* Corporation.
{*
{* DIGITAL assumes no responsibility for the use or reliability of its
{* software on equipment which is not supplied by DIGITAL.
{*
{* *****
{
{++
{
{ Facility:
{
{     VAXELN Run-time System
{
{
{ ABSTRACT:
{
{     This module contains the line printer driver for the LPV11
{     line printer interface.
```

DEVICE DRIVER EXAMPLES

```
{
{ AUTHOR:
{
{      Kris K. Barker 16-December-1983
{
{ VERSION:
{
{      V1.0-00
{
{--}
```

```
include $dap, $terminal, $kernelmsg;
```

```
const
```

```
  { Default printer characteristics - may be changed to suit the particular
    printer used }
```

```
  default_auto_cr = false;
  default_ff_lf   = false;
  default_npc_acc = true;
  default_wrap    = false;
  default_lc_uc   = false;
  lines_per_page  = 66;
  page_width      = 132;
```

```
  maximum_record_length = 132;
  ipl$power              = 31;
```

```
  line_feed          = 10;
  carriage_return    = 13;
```

```
type
```

```
  byte = 0..255;
  word = 0..65535;
```

```
  { CSR register layout }
```

```
  csr_register = [word] packed record
    inte : [pos(6)] boolean;      {interrupt enable}
    done  : boolean;              {done bit}
    ready : boolean;              {ready bit}
    error : [pos(15)] boolean;    {error}
  end; { record }
```

```
  { Data buffer register }
```

```
  data_buffer_register = [word] packed record
    data : string(1);             {character - 7 bits used}
  end; { record }
```

```
  { Register layout of the lpv-11 }
```

```
  lpv11_registers = packed record
```

```

        csr : csr_register;
        dbr : data_buffer_register;
    end; { record }

{ Interrupt communication region }
comm_region = record
    powerfail    : boolean;
    busy         : boolean;
    lp_error     : boolean;
end; { record }

var

    lpvll          : ^lpvll_registers;{pointer to device registers}
    lp_device      : device;           {device object}
    lp_priority    : integer;          {interrupt priority level}
    lp_region      : ^comm_region;     {interrupt communication region}

{ Use the $terminal sharable image so allocate a data structure }

    lp_structure   : terminal_data_pointer;

    printer_name   : name;             {printer name}
    printer_port   : port;             {printer driver's job port}
    connect_port   : port;             {connect port}
    controller_name : varying_string(8); {controller name string}
    universal_name  : varying_string(8); {controller name string}

    max_record_length : integer; {maximum record length}
    saved_record_attributes : dap$b_rat;{saved record attributes from open}

    status         : integer;          {status}
    dport          : port;
    console_present : boolean := false;

    carriage_location : integer := 0;

program lpvdriver;
{++
{
{ This is the main routine for the line printer driver. It sets up the printer
{ and waits to accept a circuit to receive the output message.
{
{
{--}

begin

    { Get the name of the line printer controller }

    controller_name := program_argument ( 1 );

```

DEVICE DRIVER EXAMPLES

```
{ Get the universal name of the line printer controller }

universal_name := program_argument ( 2 );

{ Create the device }

create_device ( controller_name,
                lp_device,
                region           := lp_region,
                registers        := lpvll,
                priority         := lp_priority,
                service_routine  := printer_interrupt,
                powerfail_routine := powerfail_recovery );

{ Create a name for the job's port and a connect port }

job_port ( printer_port );
create_name ( printer_name,
             controller_name + '0',
             printer_port,
             table := name$local );
create_port ( connect_port );

if universal_name <> '' then
    create_name ( printer_name,
                 universal_name + '0',
                 printer_port,
                 table := name$universal );

{ The $terminal module provides some handy code we can use (especially
  the various carriage control options on output). Set up the line
  printer to look like a terminal }

allocate_terminal_data ( lp_structure, 0, false );

{ Signal the kernel that we're done with initialization functions }

initialization_done;

{ Is there a console in the system? }

translate_name ( dport, 'CONSOLE', NAME$LOCAL, status := status );
if status = ker$_success then
    console_present := true;

{ Now wait for another job to connect a circuit for printer use }

while true do
    begin
```

```

    { Accept the circuit and go process the request }

    accept_circuit ( printer_port, connect := connect_port );
    status := dap$server ( circuit_port := connect_port,
                          open_action  := open_printer,
                          put_action   := write_printer );

    { Disconnect the circuit and wait for another one }

    disconnect_circuit ( connect_port )
end

end;

interrupt_service printer_interrupt ( lpv11 : ^lpv11_registers;
                                     comm   : ^comm_region );
{++
{ printer_interrupt - Line printer interrupt service routine
{
{   Inputs:
{
{       lpv11 - lpv-11 registers pointer
{       comm - Interrupt communication region
{
{   Outputs:
{
{       Device status saved in communication region and device object signalled.
{--}

var
    csr_response : csr_register;

begin

    { Read the CSR and return the status }

    with comm^, lpv11^ do
        begin
            csr_response := read_register ( csr );
            busy         := false;
            lp_error      := csr_response.error;
        end;

        { Signal the interrupt }

        signal_device

    end;

interrupt_service powerfail_recovery ( lpv11 : ^lpv11_registers;
                                       comm   : ^comm_region );
{++
{ powerfail_recovery - Powerfail recovery interrupt service routine

```

DEVICE DRIVER EXAMPLES

```
{
{ Inputs:
{
{      lpv11 - lpv-11 registers pointer
{      comm - Interrupt communication region
{
{ Outputs:
{
{      powerfail flag set
{--}

begin

with comm^ do
  begin

    { If the device was busy, signal the waiting process }

    if busy then
      signal_device;
      powerfail := true;
    end;

  end;

end;

function put_chars of type output_characters;
  { ( line_number      : integer;
    number_of_chars    : integer;
    var output_buffer  : string(number_of_chars) ) : boolean; }
{++
{
{ This function outputs characters to the line printer
{
{ Inputs:
{
{      number_of_chars specifies the number of characters to output
{      output_buffer is the buffer of characters to output
{
{ Outputs:
{
{      characters specified are output to the printer
{      return value is status of the output
{
{--}

var
  count : integer;
  csr   : csr_register;
  char_out : char;
  message_time : large_integer;
  a_character_output : boolean;
  skip : boolean;
```

```

begin

    count := 1;
    put_chars := true;
    message_time := time_value ( '0 00:00:10.0' );

    { Output all of the characters }

repeat

    { Disable interrupts and indicate the device is busy }

    csr := read_register ( lpv11^.csr );
    disable_interrupt ( lp_priority );
    lp_region^.busy := true;

    { As long as there are characters to output and the device is not
      busy, output a character }

    a_character_output := false;
    skip := false;
    while csr.done and (count <= number_of_chars) do
        begin

            { Get the character from the buffer }

            char_out := substr(output_buffer,count,1);

            { Check for line_feed and whether or not a carriage_return is to
              be output first. Then, only do it if it didn't just happen }

            if (not skip) and (char_out=chr(line_feed)) and
                default_auto_cr then
                begin
                    skip := true;
                    char_out := chr(carriage_return);
                end
            else
                skip := false;

            { If default is lower to upper case conversion, upper case
              all lower case characters }

            if default_lc_uc then
                if (char_out>='a') and (char_out<='z') then
                    char_out := chr( ord(char_out) - 32 );

```


DEVICE DRIVER EXAMPLES

```
{ Keep track of the carriage position }

if (char_out=chr(carriage_return)) or
  ( (char_out = chr(line_feed)) and not default_auto_cr) then
  carriage_location := 0
else
  carriage_location := carriage_location + 1;

{ Don't output control characters if non-printing character
  accept not selected }

if default_npc_acc or (ord(char_out) >= 32) then
  begin
    write_register ( lpvll^.dbr,
                     data := char_out );
    csr := read_register ( lpvll^.csr );
    a_character_output := true;
  end;
if not skip then
  count := count + 1;
end;

enable_interrupt;

{ If any characters were actually output, wait for the device }

if a_character_output then
  wait_any ( lp_device );

{ If there is an error, periodically write a message to the console and
  wait until the error clears (if there is no console, just wait
  on the device) }

while lp_region^.lp_error do
  begin
    if console_present then
      begin
        writeln ( '(7),controller_name+'0',' not ready' );
        wait_any ( lp_device, time := message_time )
      end
    else
      wait_any ( lp_device );

    if message_time >= time_value ( '0 00:05:00' ) then
      message_time := message_time + message_time;
    end
  end

until (count > number_of_chars);

end;
```

```

function open_printer of type dap$open_action;
{++
{ Open_action - open action routine
{
{ Inputs:      create          - create/open flag
{              file_access     - file access mode
{              share           - share access mode
{              organization     - file organization
{              record_format    - record format
{              record_attributes - record attributes
{              maximum_record_size - maximum record size
{              file_options     - file options
{              device_char      - device characteristics
{              device_depend_char - device dependent characteristics
{              file_specification - file specification
{              context          - driver specific parameter (unused)
{
{ Outputs:     organization    - file organization
{              record_format    - record format
{              record_attributes - record attributes
{              maximum_record_size - maximum record size
{              file_options     - file options
{              device_char      - device characteristics
{              device_depend_char - device dependent characteristics
{
{      return value - status
{--}

```

begin

```

    saved_record_attributes := record_attributes;

    { If no maximum record size was specified, use driver's maximum size }
    if maximum_record_size = 0 then
        maximum_record_size := maximum_record_length;

    max_record_length := maximum_record_size;

    open_printer := dap$k_success

```

end;

```

function write_printer of type dap$put_action;
{++
{ Put_action - put/write action routine
{
{ Inputs:      record_access    - record access type
{              record_number    - record number
{              record_options    - record options
{              buffer           - output buffer

```

DEVICE DRIVER EXAMPLES

```
{          buffer_length      - length of data in buffer
{          context            - driver specific parameter (unused)
{
{ Outputs:
{
{          return value - write status
{--}

var
    write_status : boolean;

begin

    write_register ( lpvll^.csr, inte := true );
    write_status := write_chars ( lp_structure,
                                buffer,
                                buffer_length,
                                saved_record_attributes,
                                put_chars );
    write_register ( lpvll^.csr, inte := false );

    write_printer := dap$k_success

end;

end; { module }
```

APPENDIX I
EXCEPTION HANDLING

I.1 EXAMPLE OF EXCEPTION HANDLING - EXCEPT_1.PAS

This example was written by the Real-time Product Support Unit of the Atlanta Customer Support Center to whom the author is grateful.

```
{ Program demonstrating the use of an exception handler           }
{                                                                 }
MODULE exception_test;                                           }

{                                                                 }
{ Include necessary utilities for the program                     }
{                                                                 }
INCLUDE $DISK_UTILITY,$FILE_UTILITY,$ELNMSG,$PASCALMSG,$KERNELMSG,
      $GET_MESSAGE_TEXT;

VAR  working_file          : TEXT;
     text_in, text_check  : VARYING_STRING(80);
     disk_label           : VARYING_STRING(12);
     pointer_to_directory : ^ELN$DIR_FILE;
     volume_name, dir_name, file_name,
     name_of_file_to_delete : VARYING_STRING(255);
     result               : INTEGER;
     which_flag           : GET_STATUS_FLAGS;
     result_text          : VARYING_STRING(255);
     SS$_INTDIV           : [EXTERNAL, VALUE] INTEGER;
```

EXCEPTION HANDLING

```
PROGRAM exception_demo(working_file, OUTPUT);
{
{ The handler is declared as a function
{
FUNCTION file_exception OF TYPE EXCEPTION_HANDLER;

{
{ Define a type to get at the severity bit
{
TYPE struct = [LONG] PACKED RECORD
    severity          : [POS(0)] 0..7;
    message_number    : [POS(3)] 0..8388607;
    facility_number    : [POS(26)] 0..3;
    cntrl              : [POS(28)] 0..15;
END;

VAR i : INTEGER;
    j : struct;

BEGIN
    WRITELN;
    WRITELN('I'm in the exception handler');
{
{ Pick up the name and severity of the exception. Then pick up the text if
{ exists.
{
    i := SIGNAL_ARGS.NAME;
    j := i::struct;
    ELN$GET_STATUS_TEXT(i, which_flag, result_text);
{
{ Change the severity to SUCCESS
{
    j.severity := 1;
    SIGNAL_ARGS.NAME::struct := j;

{ Identify the exception
{
    IF i = SS$_INTDIV THEN
        BEGIN
            WRITELN('The exception was interger divide by zero');
            WRITELN;
            file_exception := TRUE;
        END
    ELSE
{
```

```

{ If you can't handle this exception, then UNWIND the stack      }
{                                                                }
    BEGIN
        WRITELN(result_text);
        WRITELN;
        GOTO die;
    END;
END;

BEGIN
{                                                                }
{ Use the empty set to pick up all the text information, ESTABLISH the }
{ exception handler and create a directory.                          }
{                                                                }
    which_flag := [];
    ESTABLISH(file_exception);
    CREATE_DIRECTORY('DUAL:[TEMP]', result);
    WRITELN('The status for the create directory was');
{                                                                }
{ Pick up the text for the CREATE_DIRECTORY and then create a variable to }
{ be used to get the information of what files are in a directory.      }
{                                                                }
    ELN$GET_STATUS_TEXT(result, which_flag, result_text);
    WRITELN(result_text);
    WRITELN;
    NEW(pointer_to_directory);
{                                                                }
{ Write text into a file.                                              }
{                                                                }
    OPEN(working_file,
        FILE_NAME      := 'DUAL:[TEMP]TEST.TXT',
        HISTORY        := HISTORY$NEW,
        ACCESS_METHOD  := ACCESS$SEQUENTIAL,
        SHARING        := SHARE$READWRITE);
    REWRITE(working_file);
    text_in := 'This is a test';
    WRITELN(working_file, text_in);
    CLOSE(working_file);
    OPEN(working_file,
        FILE_NAME      := 'DUAL:[TEMP]TEST.TXT',
        HISTORY        := HISTORY$OLD,
        ACCESS_METHOD  := ACCESS$SEQUENTIAL,
        SHARING        := SHARE$READONLY);
    RESET(working_file);
    READLN(working_file, text_check);
    WRITELN('The text read was "', text_check, '"');
    WRITELN;

```

EXCEPTION HANDLING

```
result := 60000;
WRITELN('A integer divide by zero exception will be caused');
result := result DIV 0;
WRITELN('Now I will cause a disk related exception');
WRITELN(working_file, text_check);
die: CLOSE(working_file);
{
{ Open up the directory and get a list of the files in it.
{
  DIRECTORY_OPEN(pointer_to_directory, 'DUAL:[TEMP]TEST.TXT;', volume_name,
    dir_name, result);
  DIRECTORY_LIST(pointer_to_directory, volume_name, file_name, result);
  WRITELN('The status of the directory list was');
{
{ Pick up the text for the directory list.
{
  ELN$GET_STATUS_TEXT(result, which_flag, result_text);
  WRITELN(result_text);
{
{ Now delete the file
{
  WHILE (ODD(result)) DO
    BEGIN
      name_of_file_to_delete := volume_name + dir_name + file_name;
      WRITELN('Delete ---> ', name_of_file_to_delete);
      DELETE_FILE(name_of_file_to_delete, result);
      WRITELN('The status of the delete file was');
      ELN$GET_STATUS_TEXT(result, which_flag, result_text);
      WRITELN(result_text);
      DIRECTORY_LIST(pointer_to_directory, volume_name, file_name, result);
    END;
  END;
END;
END.
```

I.1.1.1 Running EXCEPT_1

Use the command procedure of the same name to build a system with EXCEPT_1

```

1 $      ! EXCEPT_1.COM
2 $      ! Command procedure to build the VAXELN module EXCEPT_1
3 $      ! into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG EXCEPT_1, ELN$:RTLOBJECT /LIBRARY
13 $ Link:
14 $      LINK -
15          /DEBUG EXCEPT_1 -
16          ,ELN$:RTLSHARE /LIBRARY -
17          ,ELN$:RTL /LIBRARY /INCLUDE= -
18          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
19          OTS$MSGDEF_TEXT, PASS$MSGDEF_TEXT)
20 $ System_build:
21 $      EBUILD -
22          /NOEDIT EXCEPT_1
23 $ Switch_off_verify:
24 $      SET NOVERIFY
25 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole /volumes=("DUAL SCRATCH")
2      program EXCEPT_1 /debug
3      device DUA /register=%0772150 /vector=%0154

```


APPENDIX J

EXERCISES

It is impossible to provide examples/exercises of all likely applications for VAXELN.

The practical exercises below are intended to provide practice in the use of facilities provided by VAXELN. You may wish to try working on projects of your own as well as, or instead of, those provided here. There are solutions to the exercises, in the next appendix.

The amount of language programming has been kept to a minimum deliberately. The use of a simple physics formula to generate numbers is deliberate also and the theme of temperature conversion started early in the practical work threads its way through the remaining exercises. As you proceed with the later exercises you should be able to use code written for earlier solutions.

EXERCISES

1. Write a program to provide a temperature on one of the four scales listed below given user input on one of those scales. Your program should be well laid out and use any modular techniques that are appropriate to the language that you choose to use.

Celsius	- ice melts	0.0	water boils	100.0
Fahrenheit	- ice melts	32.0	water boils	212.0
Kelvin	- ice melts	273.0	water boils	373.0
Reaumur	- ice melts	0.0	water boils	80.0

Conversion formulae:

Celsius to Fahrenheit	->	$C * 1.8 + 32.0$
Celsius to Kelvin	->	$C + 273.0$
Celsius to Reaumur	->	$C * 0.8$
Fahrenheit to Celsius	->	$F - 32.0 * 5.0 / 9.0$
Kelvin to Celsius	->	$K - 273.0$
Reaumur to Celsius	->	$R * 1.25$

EXERCISES

2. Continuing the theme of question 1 write a program that creates a subprocess to provide the conversions.
3. Write a program that creates a separate job to provide the conversions. Information about the conversion to performed should be passed to the new job via arguments in the call to CREATE_JOB.
4. Continuing the theme, use a VAXELN AREA to hold a number of temperatures e.g. about 100, and create a second job to access that area and convert the values in the area to a new scale. The old and new scale information may be held as part of a preamble to the area. Create a third job to be responsible for printing the results in the area to your terminal.
5. Write programs that use a circuit or circuits, between at least a pair of VAXELN jobs to pass temperature conversion requests and reply with the values converted.
6. Continuing the theme, write a program to read the file of temperatures TEMPS.DAT from your host VAX into an AREA. Use a second job to convert those temperatures to a new scale (TEMPS.DAT is in Celsius * 10). Write the results to the terminal.

TEMPS.DAT contains temperatures for the centre of England between 1659AD and, at the time of writing, 1985AD. Only complete years are held and there are no omissions. The data were prepared by the late Professor G. Manley with recent data by the Royal Meteorological Society and the Meteorological Office. The data look like this:

1659	30	40	60	70	110	130	160	160	130	100	50	20	88
1660	0	40	60	90	110	140	150	160	130	100	60	50	91
1661	50	50	60	80	110	140	150	150	130	110	80	60	97
1662	50	60	60	80	110	150	150	150	130	110	60	30	95
.
.
1982	26	47	59	85	115	155	165	157	141	99	78	43	97
1983	65	16	65	66	100	140	192	172	135	102	73	54	98
1984	33	32	49	82	100	147	170	179	142	116	82	55	99
1985	11	23	49	85	111	128	165	152	153	113	43	63	91

Column 1 gives the year with columns 2 through 13 providing the mean monthly temperature for each month of that year. Column 14 is the arithmetic mean of the 12 monthly values.

APPENDIX K
SOLUTIONS TO EXERCISES

K.1 SOLUTION TO EXERCISE 1 - LAB_1.PAS

```

1  {-----}
2  SOURCE:      LAB_1.PAS
3
4  PURPOSE:     To calculate temperatures on one of four scales:
5                1) Celsius      (ice melts: 0.0, water boils 100.0)
6                2) Fahrenheit   ( 32.0, 212.0)
7                3) Kelvin       (273.0, 373.0)
8                4) Reaumur      (  0.0,  80.0)
9                given an input on one of these scales.
10
11 COMPILE:     $ EPASCAL /LIST /DEBUG LAB_1
12
13 LINK:        $ LINK /DEBUG LAB_1, -
14              _$ ELN$:RTLSHARE /LIBRARY, -
15              _$ ELN$:RTL /LIBRARY
16
17 BUILD:       $ EBUILD /NOEDIT LAB_1
18
19 NOTES:       1) Command procedure LAB_1.COM compiles, links
20              and builds this module into a VAXELN system or
21              for running under VMS
22  -----}
23 MODULE Lab_1 [IDENT ('V1.000')];
24
25 CONST
26     LF      = '(10);    {line-feed}
27
28 VAR
29     User_continues  : BOOLEAN := TRUE;
30     Temp,
31     Temp_value_in,
32     Temp_value_out  : REAL;
33
34     Answer,
35     Temp_scale_in,
36     Temp_scale_out  : CHAR;

```

SOLUTIONS TO EXERCISES

```

37
38 {----- FUNCTION DECLARATIONS ----- }
39
40 FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;
41
42 FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;
43
44     BEGIN
45         Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
46     END;
47
48 {----- }
49
50 FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;
51
52     BEGIN
53         Kelvin_to_Celsius := T - 273.0;
54     END;
55
56 {----- }
57
58 FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;
59
60     BEGIN
61         Reaumur_to_Celsius := T * 1.25;
62     END;
63
64 {----- }
65
66 FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;
67
68     BEGIN
69         Celsius_to_Fahrenheit := T * 1.8 + 32.0;
70     END;
71
72 {----- }
73
74 FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;
75
76     BEGIN
77         Celsius_to_Kelvin := T + 273.0;
78     END;
79
80 {----- }
81
82 FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;
83
84     BEGIN
85         Celsius_to_Reaumur := T * 0.8;
86     END;
87
88 {----- END FUNCTION DECLARATIONS ----- }

```

```

89
90 {----- PROGRAM BLOCK -----}
91
92 PROGRAM Lab_1 (OUTPUT);
93
94     PROCEDURE Ensure_uppercase ( VAR C : CHAR );
95
96     BEGIN
97         IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
98     END {PROCEDURE Ensure_uppercase};
99
100 BEGIN
101
102 WHILE User_continues DO
103
104 BEGIN
105     WRITELN ( LF );
106
107     WRITE ( 'Please enter temperature: ' );
108     READLN ( Temp_value_in );
109
110     WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
111     READLN ( Temp_scale_in );
112     Ensure_uppercase ( Temp_scale_in );
113
114     WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
115     READLN ( Temp_scale_out );
116     Ensure_uppercase ( Temp_scale_out );
117
118 { convert everything to Celsius (or leave it in Celsius) }
119
120 CASE Temp_scale_in OF
121     'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
122
123     'R' : Temp := Reaumur_to_Celsius      ( Temp_value_in );
124
125     'K' : Temp := Kelvin_to_Celsius      ( Temp_value_in );
126
127     'C' : Temp := Temp_value_in;
128 END; {CASE Temp_scale_in}
129
130 { provide conversion }
131
132 CASE Temp_scale_out OF
133     'C' : Temp_value_out := Temp;
134
135     'F' : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
136
137     'R' : Temp_value_out := Celsius_to_Reaumur    ( Temp );
138
139     'K' : Temp_value_out := Celsius_to_Kelvin    ( Temp );
140 END; {CASE temp_scale_out}

```

SOLUTIONS TO EXERCISES

```
141
142 WRITELN ( 'Temperature before conversion: ',
143           Temp_value_in :6:1,
144           ' deg',
145           Temp_scale_in );
146
147 WRITELN ( 'Temperature after conversion: ',
148           Temp_value_out :6:1,
149           ' deg',
150           Temp_scale_out );
151
152 WRITE ( 'Do you wish to try again? [Y/N]: ' );
153 READLN ( Answer );
154
155 User_continues := Answer IN ['Y', 'y'];
156
157 END;
158
159 END {of PROGRAM}.
160 END {of MODULE };
```

K.1.1.1 Running LAB_1

```

1  $      ! LAB_1.COM
2  $      ! Command procedure to build the VAXELN
3  $      ! module LAB_1
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $ !
8  $      INQUIRE -
9          /NOPUNCTUATION  Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile
12 $ !
13 $      SET VERIFY
14 $      EPASCAL /LIST LAB_1
15 $      LINK LAB_1
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile:
19 $          @ELN_COMPILE_1  LAB_1
20 $ Link:
21 $          @ELN_LINK_1    LAB_1
22 $ System_build:
23 $          @ELN_EBUILD_1  LAB_1
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole
2      program LAB_1 /debug

```


SOLUTIONS TO EXERCISES

Output from LAB_1:

```
SH SYS
! Available: Pages: 17868, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:11.40 Total:  0 00:00:11.55
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program LAB_1, priority 16 is waiting.
!
SET TIME '3-MAR-1987 08:01:35.00'
GO
! Job 5, process 1, program LAB_1 running.
!
!Please enter temperature: 34.5
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: C
!Temperature before conversion:  34.5 degF
!Temperature after conversion:    1.4 degC
!Do you wish to try again? [Y/N]: y
!
!Please enter temperature: 212.0
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: R
!Temperature before conversion:  212.0 degF
!Temperature after conversion:    80.0 degR
!Do you wish to try again? [Y/N]: y
!
!Please enter temperature: -40
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: F
!Temperature before conversion:  -40.0 degC
!Temperature after conversion:   -40.0 degF
!Do you wish to try again? [Y/N]: y
!
!Please enter temperature: -35.2
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: K
!Temperature before conversion:  -35.2 degC
!Temperature after conversion:   237.8 degK
!Do you wish to try again? [Y/N]: n
! Job 5, process 1, program LAB_1 has exited.
!
exit
```

K.2 SOLUTION TO EXERCISE 1 - LAB_1_F.FOR

```

1  * SOURCE:           LAB_1_F.FOR
2  *
3  * PURPOSE:          To calculate temperatures on one of four scales:
4  *                   1) Celsius      (ice melts: 0.0, water boils 100.0)
5  *                   2) Fahrenheit  ( 32.0, 212.0)
6  *                   3) Kelvin      (273.0, 373.0)
7  *                   4) Reaumur     ( 0.0, 80.0)
8  *                   given an input on one of these scales.
9  *
10 * COMPILER:         $ FORTRAN /LIST /DEBUG /NOOPTIMIZE LAB_1_F
11 *
12 * LINK:             $ LINK /DEBUG /NOSYSLIB LAB_1_F, -
13 *                   _$ ELN$:FRTLOBJECT /LIBRARY, -
14 *                   _$ ELN$:RTLSHARE /LIBRARY, -
15 *                   _$ ELN$:RTL /LIBRARY
16 *
17 * BUILD:            $ EBUILD /NOEDIT LAB_1_F
18 *
19 * NOTES:            1) Command procedure LAB_1_F.COM compiles, links
20 *                   and builds this module into a VAXELN system or
21 *                   for running under VMS
22 *
23
24     IMPLICIT NONE
25
26     LOGICAL          User_continues /.TRUE./
27
28     REAL             Temp,
29     1                Temp_value_in,
30     1                Temp_value_out,
31     1                Fahrenheit_to_Celsius,
32     1                Reaumur_to_Celsius,
33     1                Kelvin_to_Celsius,
34     1                Celsius_to_Fahrenheit,
35     1                Celsius_to_Kelvin,
36     1                Celsius_to_Reaumur
37
38     CHARACTER        Answer,
39     1                Temp_scale_in,
40     1                Temp_scale_out
41

```

SOLUTIONS TO EXERCISES

```

42      DO WHILE (User_continues)
43
44      WRITE ( 6, '(/1X, 'Please enter temperature: ' $)')
45      READ  ( 5, * ) Temp_value_in
46
47      WRITE ( 6, '(1X, 'Please enter scale for that ',
48      1      'temperature [K,C,F,R]: ' $)')
49      READ  ( 5, '(A)' ) Temp_scale_in
50
51      * convert scale to uppercase if necessary
52      CALL LC2UC ( Temp_scale_in, Temp_scale_in )
53
54      WRITE ( 6, '(1X, 'Please enter the conversion ',
55      1      'required [K,C,F,R]: ' $)')
56      READ  ( 5, '(A)' ) Temp_scale_out
57
58      * convert scale to uppercase if necessary
59      CALL LC2UC ( Temp_scale_out, Temp_scale_out )
60
61      * convert everything to Celsius or leave it in Celsius
62
63      IF (Temp_scale_in .EQ. 'F')
64      1      Temp = Fahrenheit_to_Celsius ( Temp_value_in )
65
66      IF (Temp_scale_in .EQ. 'R')
67      1      Temp = Reaumur_to_Celsius ( Temp_value_in )
68
69      IF (Temp_scale_in .EQ. 'K')
70      1      Temp = Kelvin_to_Celsius ( Temp_value_in )
71
72      IF (Temp_scale_in .EQ. 'C')
73      1      Temp = Temp_value_in
74
75
76      * provide conversion *
77
78      IF (Temp_scale_out .EQ. 'C')
79      1      Temp_value_out = Temp
80
81      IF (Temp_scale_out .EQ. 'F')
82      1      Temp_value_out = Celsius_to_Fahrenheit ( Temp )
83
84      IF (Temp_scale_out .EQ. 'R')
85      1      Temp_value_out = Celsius_to_Reaumur ( Temp )
86
87      IF (Temp_scale_out .EQ. 'K')
88      1      Temp_value_out = Celsius_to_Kelvin ( Temp )

```

```

89
90     WRITE ( 6, '(1X, ''Temperature before conversion: '',
91             1          F6.1, '' deg'', A )')
92     1      Temp_value_in, Temp_scale_in
93
94     WRITE ( 6, '(1X, ''Temperature after conversion: '',
95             1          F6.1, '' deg'', A )')
96     1      Temp_value_out, Temp_scale_out
97
98     WRITE ( 6, '(1X, ''Do you wish to try again? [Y/N]: '' $)')
99     READ  ( 5, '(A)') Answer
100
101     User_continues = Answer .EQ. 'Y' .OR. Answer .EQ. 'y'
102
103     END DO
104
105     END
106
107 * -----
108
109     REAL FUNCTION Fahrenheit_to_Celsius (T)
110
111     IMPLICIT NONE
112     REAL    T
113
114     Fahrenheit_to_Celsius = (T - 32.0) * 5.0 / 9.0
115
116     RETURN
117     END
118
119 * -----
120
121     REAL FUNCTION Kelvin_to_Celsius (T)
122
123     IMPLICIT NONE
124     REAL    T
125
126     Kelvin_to_Celsius = T - 273.0
127
128     RETURN
129     END
130

```

SOLUTIONS TO EXERCISES

```
131  * -----
132
133      REAL FUNCTION Reaumur_to_Celsius (T)
134
135      IMPLICIT NONE
136      REAL    T
137
138      Reaumur_to_Celsius = T * 1.25
139
140      RETURN
141      END
142
143  * -----
144
145      REAL FUNCTION Celsius_to_Fahrenheit (T)
146
147      IMPLICIT NONE
148      REAL    T
149
150      Celsius_to_Fahrenheit = T * 1.8 + 32.0
151
152      RETURN
153      END
154
155  * -----
156
157      REAL FUNCTION Celsius_to_Kelvin (T)
158
159      IMPLICIT NONE
160      REAL    T
161
162      Celsius_to_Kelvin = T + 273.0
163
164      RETURN
165      END
166
167  * -----
168
169      REAL FUNCTION Celsius_to_Reaumur (T)
170
171      IMPLICIT NONE
172      REAL    T
173
174      Celsius_to_Reaumur = T * 0.8
175
176      RETURN
177      END
178
```

```

179 * -----
180
181     SUBROUTINE LC2UC (INSTR,OUTSTR)
182 *
183 *     Subroutine to convert lowercase letters to uppercase
184 *
185 *     Input required is string and output location
186 *
187     INTEGER          K,
188     1                Key
189     CHARACTER        INSTR*(*),
190     1                OUTSTR*(*),
191     1                Lcase *26,
192     1                Ucase *26
193
194     DATA Lcase /'abcdefghijklmnopqrstuvwxyz'/
195     DATA Ucase /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
196
197     DO K = 1, LEN(INSTR), 1
198
199         Key = INDEX (Lcase, Instr(K:K))
200
201         IF (Key .GT. 0) THEN
202             Outstr(K:K) = Ucase(Key:Key)
203         ELSE
204             Outstr(K:K) = Instr(K:K)
205         END IF
206
207     END DO
208
209     RETURN
210     END

```

SOLUTIONS TO EXERCISES

K.2.1 Running LAB_1_F.FOR

Use the command procedure of the same name to build a system with LAB_1_F:

```
1 $      ! LAB_1_F.COM
2 $      ! Command procedure to compile and link the VAXELN
3 $      ! module LAB_1_F
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      INQUIRE -
9          /NOPUNCTUATION Running_under_VMS -
10         "Do you wish to run this program under VMS [Y/N]: "
11 $      IF .NOT. Running_under_VMS THEN GOTO Compile
12 $ !
13 $      SET VERIFY
14 $      FORTRAN /LIST LAB_1_F
15 $      LINK LAB_1_F
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile:
19 $          FORTRAN -
20              /LIST -
21              /DEBUG -
22              /NOOPTIMIZE LAB_1_F
23 $ Link:
24 $          LINK -
25              /NOSYSLIB -
26              /DEBUG LAB_1_F -
27              ,ELN$:FRTLOBJECT /LIBRARY -
28              ,RTLSHARE /LIBRARY -
29              ,RTL /LIBRARY
30 $      SET NOVERIFY
31 $ System_build:
32 $          @ELN_EBUILD_1 LAB_1_F
33 $ Switch_off_verify:
34 $      SET NOVERIFY
35 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_1_F /debug
```

Output from LAB_1_F:

```

SH SYS
! Available: Pages: 17824, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:14.25 Total:  0 00:00:14.41
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSESERVER, priority 16 is waiting.
! Job 5, program LAB_1_F, priority 16 is waiting.
!
GO
! Job 5, process 1, program LAB_1_F running.
!
!Please enter temperature: 12.4
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: F
!Temperature before conversion:  12.4 degC
!Temperature after conversion:   54.3 degF
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 80
!Please enter scale for that temperature [K,C,F,R]: R
!Please enter the conversion required [K,C,F,R]: c
!Temperature before conversion:  80.0 degR
!Temperature after conversion:  100.0 degC
!Do you wish to try again? [Y/N]: y
!
!Please enter temperature: 98.4
!Please enter scale for that temperature [K,C,F,R]: f
!Please enter the conversion required [K,C,F,R]: k
!Temperature before conversion:  98.4 degF
!Temperature after conversion:  309.9 degK
!Do you wish to try again? [Y/N]: y
!
!Please enter temperature: 0.0
!Please enter scale for that temperature [K,C,F,R]: f
!Please enter the conversion required [K,C,F,R]: c
!Temperature before conversion:  0.0 degF
!Temperature after conversion:  -17.8 degC
!Do you wish to try again? [Y/N]: n
! Job 5, process 1, program LAB_1_F has exited.
!
exit

```


SOLUTIONS TO EXERCISES

K.3 SOLUTION TO EXERCISE 1 - LAB_1_C.C

```
1  /* -----
2  * SOURCE:          LAB_1_C.C
3  *
4  * PURPOSE:        To calculate temperatures on one of four scales:
5  *                  1) Celsius      (ice melts: 0.0, water boils 100.0)
6  *                  2) Fahrenheit  ( 32.0, 212.0)
7  *                  3) Kelvin      (273.0, 373.0)
8  *                  4) Reaumur     (  0.0,  80.0)
9  *                  given an input on one of these scales.
10 *
11 * COMPILE:        $ CC /LIST /DEBUG LAB_1_C + ELN$:VAXELNC /LIBRARY
12 *
13 * LINK:           $ LINK /DEBUG LAB_1_C, -
14 *                  _$ ELN$:CRTLSHARE /LIBRARY, -
15 *                  _$ ELN$:RTLSHARE /LIBRARY, -
16 *                  _$ ELN$:RTL /LIBRARY
17 *
18 * BUILD:          $ EBUILD /NOEDIT LAB_1_C
19 *
20 * NOTES:          1) Command procedure LAB_1_C.COM compiles, links
21 *                  and builds this module into a VAXELN system or
22 *                  for running under VMS
23 * -----
24 */
25
26 #include $vaxelnc
27 #include ctype      /* for the toupper function */
28
29 BOOLEAN user_continues = TRUE;
30
31 float   temp,
32         temp_value_in,
33         temp_value_out;
34
35 char    answer,
36         temp_scale_in,
37         temp_scale_out;
38
```

```

39 lab_1 ()
40     MAIN_PROGRAM
41 {
42     float    fahrenheit_to_celsius(),
43             kelvin_to_celsius(),
44             reaumur_to_celsius(),
45             celsius_to_fahrenheit(),
46             celsius_to_kelvin(),
47             celsius_to_reaumur();
48
49     char      dummy;
50
51     while (user_continues)
52     {
53         printf ("\n\n");
54         printf ("Please enter temperature: ");
55         scanf ("%f", &temp_value_in);
56
57         dummy = getchar();    /* dispose of end-of-line */
58         printf ("\nPlease enter scale for that temperature [K,C,F,R]: ");
59         scanf ("%c", &temp_scale_in);
60
61         /*      uppercase temp_scale_in as necessary
62          */
63         temp_scale_in = toupper (temp_scale_in);
64
65         dummy = getchar();    /* dispose of end-of-line */
66         printf ("Please enter the conversion required [K,C,F,R]: ");
67         scanf ("%c", &temp_scale_out);
68
69         /*      uppercase temp_scale_out as necessary
70          */
71         temp_scale_out = toupper (temp_scale_out);
72
73         dummy = getchar();    /* dispose of end-of-line */
74
75         switch (temp_scale_in )
76         {
77             case 'F': temp = fahrenheit_to_celsius (temp_value_in);
78                     break;
79             case 'R': temp = reaumur_to_celsius      (temp_value_in);
80                     break;
81             case 'K': temp = kelvin_to_celsius      (temp_value_in);
82                     break;
83             case 'C': temp = temp_value_in;
84                     break;
85             } /* switch temp_scale_in */
86

```

SOLUTIONS TO EXERCISES

```

87  switch (temp_scale_out)
88      {
89      case 'C': temp_value_out = temp;
90              break;
91      case 'F': temp_value_out = celsius_to_fahrenheit (temp);
92              break;
93      case 'R': temp_value_out = celsius_to_reaumur (temp);
94              break;
95      case 'K': temp_value_out = celsius_to_kelvin (temp);
96              break;
97      } /* switch temp_scale_out */
98
99  printf ("Temperature before conversion: %6.1f deg%c\n",
100         temp_value_in, temp_scale_in);
101  printf ("Temperature after conversion: %6.1f deg%c\n",
102         temp_value_out, temp_scale_out);
103
104  printf ("Do you wish to try again? [Y/N]: ");
105  scanf ("%c", &answer);
106
107      user_continues = (answer == 'Y' || answer == 'y');
108
109      } /* while user continues */
110
111  }
112
113  /*  ===== Temperature conversion functions =====
114      */
115
116  float fahrenheit_to_celsius (t)
117  float t;
118      {
119      return ((t-32.0) * 5.0 / 9.0);
120      }
121

```

```
122 /* ----- */
123
124 float kelvin_to_celsius (t)
125 float t;
126     {
127         return (t - 273.0);
128     }
129
130 /* ----- */
131
132 float reaumur_to_celsius (t)
133 float t;
134     {
135         return (t * 1.25);
136     }
137
138 /* ----- */
139
140 float celsius_to_fahrenheit (t)
141 float t;
142     {
143         return (t * 1.8 + 32.0);
144     }
145
146 /* ----- */
147
148 float celsius_to_kelvin (t)
149 float t;
150     {
151         return (t + 273.0);
152     }
153
154 /* ----- */
155
156 float celsius_to_reaumur (t)
157 float t;
158     {
159         return (t * 0.8);
160     }
```

SOLUTIONS TO EXERCISES

K.3.1 Running LAB_1_C

```
1 $      ! LAB_1_C.COM
2 $      ! Command procedure to build the VAXELN module LAB_1_C
3 $      !
4 $      ON ERROR THEN GOTO Switch_off_verify
5 $      SET DEFAULT Default_directory
6 $ !
7 $      INQUIRE -
8          /NOPUNCTUATION Running_under_VMS -
9          "Do you wish to run this program under VMS [Y/N]: "
10 $      IF .NOT. Running_under_VMS THEN GOTO Compile_LAB_1_C
11 $ !
12 $      SET VERIFY
13 $      CC /LIST LAB_1_C + ELN$:VAXELNC /LIBRARY
14 $      LINK LAB_1_C, SYS$INPUT /OPTION
15 SYS$SHARE:VAXCRTL/SHARE
16 $      GOTO Switch_off_verify
17 $ !
18 $ Compile_LAB_1_C:
19 $      SET VERIFY
20 $      CC -
21          /LIST -
22          /DEBUG LAB_1_C + ELN$:VAXELNC /LIBRARY
23 $ Link_LAB_1_C:
24 $      LINK -
25          /DEBUG LAB_1_C -
26          , ELN$:CRTLSHARE /LIBRARY -
27          ,      RTLSHARE /LIBRARY -
28          ,      RTL      /LIBRARY
29 $      SET NOVERIFY
30 $ System_build:
31 $      @ELN_EBUILD_1 LAB_1_C
32 $ Switch_off_verify:
33 $      SET NOVERIFY
34 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_1_C /debug
```

Output from LAB_1_C:

```

SH SYS
! Available: Pages: 1391, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:12.48 Total:  0 00:00:12.62
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program LAB_1_C, priority 16 is waiting.
!
GO
! Job 5, process 1, program LAB_1_C running.
!
!Please enter temperature: 32.0
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: C
!Temperature before conversion:  32.0 degF
!Temperature after conversion:    0.0 degC
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 23
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: F
!Temperature before conversion:  23.0 degC
!Temperature after conversion:   73.4 degF
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 373
!Please enter scale for that temperature [K,C,F,R]: K
!Please enter the conversion required [K,C,F,R]: F
!Temperature before conversion:  373.0 degK
!Temperature after conversion:   212.0 degF
!Do you wish to try again? [Y/N]: N
! Job 5, process 1, program LAB_1_C has exited.
!
EXIT

```

SOLUTIONS TO EXERCISES

K.4 SOLUTION TO EXERCISE 2 - LAB_2.PAS

```
1  {-----}
2  SOURCE:      LAB_2.PAS
3
4  PURPOSE:      To calculate temperatures on one of four scales:
5                  1) Celsius      (ice melts: 0.0, water boils 100.0)
6                  2) Fahrenheit   ( 32.0, 212.0)
7                  3) Kelvin       (273.0, 373.0)
8                  4) Reaumur      (  0.0,  80.0)
9                  given an input on one of these scales.
10                 A subprocess is created to perform the conversion.
11
12  COMPILE:      $ EPASCAL /LIST /DEBUG LAB_2
13
14  LINK:          $ LINK /DEBUG LAB_2, -
15                  _$ ELN$:RTLSHARE /LIBRARY, -
16                  _$ ELN$:RTL /LIBRARY
17
18  BUILD:         $ EBUILD /NOEDIT LAB_2
19
20  NOTES:         1) Command procedure LAB_2.COM compiles, links
21                  and builds this module into a VAXELN system
22  -----}
23  MODULE Lab_2 [IDENT ('V1.000')];
24
25
26  CONST
27      LF      = '(10);  {line-feed}
28
29  VAR
30      User_continues  : BOOLEAN := TRUE;
31
32      Temp,
33      Temp_value_in,
34      Temp_value_out  : REAL;
35
36      Sub_proc_exit_status,
37      Status_returned : INTEGER;
38
39      Answer,
40      Temp_scale_in,
41      Temp_scale_out  : CHAR;
42
43      Process_ID      : PROCESS;
44
```

```

45 {----- FUNCTION DECLARATIONS ----- }
46
47 FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;
48
49 FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;
50
51     BEGIN
52         Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
53     END;
54
55 {----- }
56
57 FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;
58
59     BEGIN
60         Kelvin_to_Celsius := T - 273.0;
61     END;
62
63 {----- }
64
65 FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;
66
67     BEGIN
68         Reaumur_to_Celsius := T * 1.25;
69     END;
70
71 {----- }
72
73 FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;
74
75     BEGIN
76         Celsius_to_Fahrenheit := T * 1.8 + 32.0;
77     END;
78
79 {----- }
80
81 FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;
82
83     BEGIN
84         Celsius_to_Kelvin := T + 273.0;
85     END;
86
87 {----- }
88
89 FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;
90
91     BEGIN
92         Celsius_to_Reaumur := T * 0.8;
93     END;
94
95 {----- END FUNCTION DECLARATIONS ----- }

```


SOLUTIONS TO EXERCISES

```

96
97 PROCESS_BLOCK Temperature_server;
98
99 BEGIN
100
101 CASE Temp_scale_in OF
102     'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
103
104     'R' : Temp := Reaumur_to_Celsius      ( Temp_value_in );
105
106     'K' : Temp := Kelvin_to_Celsius      ( Temp_value_in );
107
108     'C' : Temp := Temp_value_in;
109 END; {CASE Temp_scale_in}
110
111 { provide conversion }
112
113 CASE Temp_scale_out OF
114     'C' : Temp_value_out := Temp;
115
116     'F' : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
117
118     'R' : Temp_value_out := Celsius_to_Reaumur    ( Temp );
119
120     'K' : Temp_value_out := Celsius_to_Kelvin     ( Temp );
121 END; {CASE temp_scale_out}
122
123 END { end of PROCESS Temperature_server };
124
125 {----- PROGRAM BLOCK -----}
126
127 PROGRAM Lab_2 (OUTPUT);
128
129     PROCEDURE Ensure_uppercase ( VAR C : CHAR );
130
131     BEGIN
132         IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
133     END {PROCEDURE Ensure_uppercase};
134
135 BEGIN
136

```

```

137 WHILE User_continues DO
138
139 BEGIN
140     WRITELN ( LF );
141
142     WRITE ( 'Please enter temperature: ' );
143     READLN ( Temp_value_in );
144
145     WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
146     READLN ( Temp_scale_in );
147     Ensure_uppercase ( Temp_scale_in );
148
149     WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
150     READLN ( Temp_scale_out );
151     Ensure_uppercase ( Temp_scale_out );
152
153     CREATE_PROCESS ( Process_ID,
154                     Temperature_server,
155                     EXIT    := Sub_proc_exit_status,
156                     STATUS  := Status_returned );
157
158     IF NOT ODD ( Status_returned ) THEN
159         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
160
161     WAIT_ANY ( Process_ID,
162              STATUS := Status_returned );
163     IF NOT ODD ( Status_returned ) THEN
164         WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);
165
166     IF NOT ODD ( Sub_proc_exit_status ) THEN
167         WRITELN ( 'Subprocess exit status was: ', Status_returned :1);
168
169     WRITELN ( 'Temperature before conversion: ',
170              Temp_value_in :6:1,
171              ' deg',
172              Temp_scale_in );
173
174     WRITELN ( 'Temperature after conversion: ',
175              Temp_value_out :6:1,
176              ' deg',
177              Temp_scale_out );
178
179     WRITE ( 'Do you wish to try again? [Y/N]: ' );
180     READLN ( Answer );
181
182     User_continues := Answer IN ['Y', 'y'];
183
184 END;
185
186 END {of PROGRAM}.
187 END {of MODULE };

```

SOLUTIONS TO EXERCISES

K.4.1 Running LAB_2

```
1 $      ! LAB_2.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module LAB_2
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $          @ELN_COMPILE_1  LAB_2
10 $ Link:
11 $          @ELN_LINK_1    LAB_2
12 $ System_build:
13 $          @ELN_EBUILD_1  LAB_2
14 $ Switch_off_verify:
15 $          SET NOVERIFY
16 $          EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_2 /debug
```

Output from LAB_2:

SH SYS

! Available: Pages: 17869, Page table slots: 51, Pool blocks: 274
! Time since SET TIME: Idle: 0 00:00:52.39 Total: 0 00:00:52.55
! Time used by past jobs: 0 00:00:00.02

!

! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program LAB_2, priority 16 is waiting.

!

GO

! Job 5, process 1, program LAB_2 running.

!

!Please enter temperature: 34.5
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: K

!

! Job 5, process 2, program LAB_2 needs attention.

! Module LAB_2

! 98:

!>>99: BEGIN

! 100:

! 101: CASE Temp_scale_in OF

! 102: 'F' : Temp := Fahrenheit_to_Celsius (Temp_value_in);

! 103:

GO

! Job 5, process 2, program LAB_2 running.

! Job 5, process 2, program LAB_2 has exited.

!

! Job 5, process 1, program LAB_2 running.

!Temperature before conversion: 34.5 degC

!Temperature after conversion: 307.5 degK

!Do you wish to try again? [Y/N]: Y

!

!Please enter temperature: 32.0

!Please enter scale for that temperature [K,C,F,R]: F

!Please enter the conversion required [K,C,F,R]: C

SOLUTIONS TO EXERCISES

```
!  
! Job 5, process 3, program LAB_2 needs attention.  
! Module LAB_2  
! 98:  
!>>99: BEGIN  
! 100:  
! 101: CASE Temp_scale_in OF  
! 102:      'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );  
! 103:  
GO  
! Job 5, process 3, program LAB_2 running.  
! Job 5, process 3, program LAB_2 has exited.  
!  
! Job 5, process 1, program LAB_2 running.  
!Temperature before conversion: 32.0 degF  
!Temperature after conversion: 0.0 degC  
!Do you wish to try again? [Y/N]: N  
!  
! Job 5, process 1, program LAB_2 has exited.  
!  
EXIT
```

K.5 SOLUTION TO EXERCISE 2 - LAB_2A.PAS

```

1  {-----}
2  SOURCE:      LAB_2A.PAS
3
4  PURPOSE:      To calculate temperatures on one of four scales:
5                  1) Celsius      (ice melts: 0.0, water boils 100.0)
6                  2) Fahrenheit   ( 32.0, 212.0)
7                  3) Kelvin       (273.0, 373.0)
8                  4) Reaumur      (  0.0,  80.0)
9                  given an input on one of these scales.
10                 This version INCLUDEs the definitions of the temperature
11                 conversion functions.
12
13  COMPILE:      $ EPASCAL /LIST /DEBUG LAB_2A_TDEFS
14                 $ EPASCAL /LIST /DEBUG LAB_2A, LAB_2A_TDEFS /MODULE
15
16  LINK:         $ LINK /DEBUG LAB_2A, LAB_2A_TDEFS, -
17                 _$ ELN$:RTLSHARE /LIBRARY, -
18                 _$ ELN$:RTL /LIBRARY
19
20  BUILD:        $ EBUILD /NOEDIT LAB_2A
21
22  NOTES:        1) Command procedure LAB_2A.COM compiles, links
23                 and builds this module into a VAXELN system
24  -----}
25  MODULE Lab_2A [IDENT ('V1.000')];
26
27  INCLUDE
28      Lab_2A_Tdefs; { declarations of temperature functions }
29
30  CONST
31      LF      = '(10); {line-feed}
32
33  VAR
34      User_continues : BOOLEAN := TRUE;
35
36      Temp,
37      Temp_value_in,
38      Temp_value_out : REAL;
39
40      Sub_proc_exit_status,
41      Status_returned : INTEGER;
42
43      Answer,
44      Temp_scale_in,
45      Temp_scale_out : CHAR;
46
47      Process_ID      : PROCESS;
48

```

SOLUTIONS TO EXERCISES

```

49 { ----- }
50
51 PROCESS_BLOCK Temperature_server;
52
53 BEGIN
54
55 CASE Temp_scale_in OF
56     'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
57
58     'R' : Temp := Reaumur_to_Celsius      ( Temp_value_in );
59
60     'K' : Temp := Kelvin_to_Celsius      ( Temp_value_in );
61
62     'C' : Temp := Temp_value_in;
63 END; {CASE Temp_scale_in}
64
65 { provide conversion }
66
67 CASE Temp_scale_out OF
68     'C' : Temp_value_out := Temp;
69
70     'F' : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
71
72     'R' : Temp_value_out := Celsius_to_Reaumur     ( Temp );
73
74     'K' : Temp_value_out := Celsius_to_Kelvin      ( Temp );
75 END; {CASE temp_scale_out}
76
77 END { end of PROCESS Temperature_server };
78

```

```

79  {----- PROGRAM BLOCK -----}
80
81  PROGRAM Lab_2A (OUTPUT);
82
83      PROCEDURE Ensure_uppercase ( VAR C : CHAR );
84
85      BEGIN
86          IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
87      END {PROCEDURE Ensure_uppercase};
88
89  BEGIN
90
91  WHILE User_continues DO
92
93  BEGIN
94      WRITELN ( LF );
95
96      WRITE ( 'Please enter temperature: ' );
97      READLN ( Temp_value_in );
98
99      WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
100     READLN ( Temp_scale_in );
101     Ensure_uppercase ( Temp_scale_in );
102
103     WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
104     READLN ( Temp_scale_out );
105     Ensure_uppercase ( Temp_scale_out );
106
107     CREATE_PROCESS ( Process_ID,
108                     Temperature_server,
109                     EXIT    := Sub_proc_exit_status,
110                     STATUS  := Status_returned );
111
112     IF NOT ODD ( Status_returned ) THEN
113         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
114
115     WAIT_ANY ( Process_ID,
116              STATUS := Status_returned );
117     IF NOT ODD ( Status_returned ) THEN
118         WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);
119
120     IF NOT ODD ( Sub_proc_exit_status ) THEN
121         WRITELN ( 'Subprocess exit status was: ', Status_returned :1);
122

```


SOLUTIONS TO EXERCISES

```
123  WRITELN (    'Temperature before conversion: ',
124              Temp_value_in :6:1,
125              ' deg',
126              Temp_scale_in );
127
128  WRITELN (    'Temperature after conversion: ',
129              Temp_value_out :6:1,
130              ' deg',
131              Temp_scale_out );
132
133  WRITE ( 'Do you wish to try again? [Y/N]: ' );
134  READLN ( Answer );
135
136  User_continues := Answer IN ['Y', 'y'];
137
138  END;
139
140  END {of PROGRAM}.
141  END {of MODULE };
```

K.6 SOLUTION TO EXERCISE 2 - LAB_2A_TDEFS.PAS

The file of included definitions looks like this:

```

1  MODULE Lab_2A_Tdefs;
2
3  { Temperature conversion functions }
4
5  FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;
6
7  FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;
8
9      BEGIN
10         Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
11     END;
12
13 {----- }
14
15 FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;
16
17     BEGIN
18         Kelvin_to_Celsius := T - 273.0;
19     END;
20
21 {----- }
22
23 FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;
24
25     BEGIN
26         Reaumur_to_Celsius := T * 1.25;
27     END;
28
29 {----- }
30
31 FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;
32
33     BEGIN
34         Celsius_to_Fahrenheit := T * 1.8 + 32.0;
35     END;
36

```

SOLUTIONS TO EXERCISES

```
37 {----- }
38
39 FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;
40
41     BEGIN
42         Celsius_to_Kelvin := T + 273.0;
43     END;
44
45 {----- }
46
47 FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;
48
49     BEGIN
50         Celsius_to_Reaumur := T * 0.8;
51     END;
52
53 {----- END FUNCTION DECLARATIONS ----- }
54
55 END.
```

K.6.1 Running LAB_2A

```
1 $      ! LAB_2A.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module LAB_2A
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG LAB_2A_TDEFS
13 $      EPASCAL -
14          /LIST -
15          /DEBUG LAB_2A, LAB_2A_TDEFS /MODULE
16 $ Link:
17 $      LINK -
18          /DEBUG LAB_2A, LAB_2A_TDEFS -
19          ,ELN$:RTLSHARE /LIBRARY -
20          ,RTL /LIBRARY
21 $      SET NOVERIFY
22 $ System_build:
23 $      @ELN_EBUILD_1 LAB_2A
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_2A /debug
```

SOLUTIONS TO EXERCISES

K.7 SOLUTION TO EXERCISE 2 - LAB_2B.PAS

This version INCLUDEs the declarations and constants.

```

1  {-----}
2  SOURCE:      LAB_2B.PAS
3
4  PURPOSE:     To calculate temperatures on one of four scales:
5                1) Celsius      (ice melts: 0.0, water boils 100.0)
6                2) Fahrenheit   ( 32.0, 212.0)
7                3) Kelvin       (273.0, 373.0)
8                4) Reaumur      (  0.0,  80.0)
9                given an input on one of these scales.
10               This version uses a subprocess to perform the
11               conversions and INCLUDEs the definitions of the
12               temperature conversion functions, constants,
13               variables and process block.
14
15  COMPILER:    $ EPASCAL /LIST /DEBUG LAB_2B_DEFS
16               $ EPASCAL /LIST /DEBUG LAB_2B, LAB_2B_DEFS /MODULE
17
18  LINK:        $ LINK /DEBUG LAB_2B, LAB_2B_DEFS, -
19               _$ ELN$:RTLSHARE /LIBRARY, -
20               _$ ELN$:RTL /LIBRARY
21
22  BUILD:       $ EBUILD /NOEDIT LAB_2B
23
24  NOTES:       1) Command procedure LAB_2B.COM compiles, links
25               and builds this module into a VAXELN system
26  -----}
27  MODULE Circuit_01 [IDENT ('V1.000')];
28
29  INCLUDE
30      LAB_2B_DEFS; { declarations of constants, variables,
31                   temperature functions, and process block }
32

```

```

33 {----- PROGRAM BLOCK -----}
34
35 PROGRAM Create_job_01 (OUTPUT);
36
37     PROCEDURE Ensure_uppercase ( VAR C : CHAR );
38
39     BEGIN
40         IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
41     END {PROCEDURE Ensure_uppercase};
42
43 BEGIN
44
45 WHILE User_continues DO
46
47 BEGIN
48     WRITELN ( LF );
49
50     WRITE ( 'Please enter temperature: ' );
51     READLN ( Temp_value_in );
52
53     WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
54     READLN ( Temp_scale_in );
55     Ensure_uppercase ( Temp_scale_in );
56
57     WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
58     READLN ( Temp_scale_out );
59     Ensure_uppercase ( Temp_scale_out );
60
61 CREATE_PROCESS ( Process_ID,
62                 Temperature_server,
63                 EXIT := Sub_proc_exit_status,
64                 STATUS := Status_returned );
65
66     IF NOT ODD ( Status_returned ) THEN
67         WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);
68
69 WAIT_ANY ( Process_ID,
70           STATUS := Status_returned );
71     IF NOT ODD ( Status_returned ) THEN
72         WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);
73
74 IF NOT ODD ( Sub_proc_exit_status ) THEN
75 WRITELN ( 'Subprocess exit status was: ', Status_returned :1);

```

SOLUTIONS TO EXERCISES

```
76
77  WRITELN (    'Temperature before conversion: ',
78              Temp_value_in :6:1,
79              ' deg',
80              Temp_scale_in );
81
82  WRITELN (    'Temperature after conversion: ',
83              Temp_value_out :6:1,
84              ' deg',
85              Temp_scale_out );
86
87  WRITE ( 'Do you wish to try again? [Y/N]: ' );
88  READLN ( Answer );
89
90  User_continues := Answer IN ['Y', 'y'];
91
92  END;
93
94  END {of PROGRAM}.
95  END {of MODULE };
```

K.8 SOLUTION TO EXERCISE 2 - LAB_2B_DEFS.PAS

The file of included definitions looks like this:

```

1  MODULE LAB_2B_DEFS;
2
3  { Temperature conversion functions }
4
5  INCLUDE
6      LAB_2A_TDEFS;
7
8  CONST
9      LF      = '(10);  {line-feed}
10
11  VAR
12      User_continues  : BOOLEAN := TRUE;
13
14      Temp,
15      Temp_value_in,
16      Temp_value_out  : REAL;
17
18      Sub_proc_exit_status,
19      Status_returned : INTEGER;
20
21      Answer,
22      Temp_scale_in,
23      Temp_scale_out  : CHAR;
24
25      Process_ID      : PROCESS;
26
27  { ----- }
28

```


SOLUTIONS TO EXERCISES

```
29  PROCESS_BLOCK Temperature_server;
30
31  BEGIN
32
33  CASE Temp_scale_in OF
34      'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
35
36      'R' : Temp := Reaumur_to_Celsius      ( Temp_value_in );
37
38      'K' : Temp := Kelvin_to_Celsius      ( Temp_value_in );
39
40      'C' : Temp := Temp_value_in;
41  END; {CASE Temp_scale_in}
42
43  { provide conversion }
44
45  CASE Temp_scale_out OF
46      'C' : Temp_value_out := Temp;
47
48      'F' : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
49
50      'R' : Temp_value_out := Celsius_to_Reaumur     ( Temp );
51
52      'K' : Temp_value_out := Celsius_to_Kelvin      ( Temp );
53  END; {CASE temp_scale_out}
54
55  END { end of PROCESS Temperature_server };
56  END { of MODULE }.
```

K.8.1 Running LAB_2B

```

1 $      ! LAB_2B.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! module LAB_2B
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG LAB_2A_TDEFS
13 $      EPASCAL -
14          /LIST -
15          /DEBUG LAB_2B_DEFS, LAB_2A_TDEFS /MODULE
16 $      EPASCAL -
17          /LIST -
18          /DEBUG LAB_2B, LAB_2B_DEFS /MODULE
19 $ Link:
20 $      LINK -
21          /DEBUG LAB_2B, LAB_2B_DEFS, LAB_2A_TDEFS -
22          ,ELN$:RTLSHARE /LIBRARY -
23          ,RTL /LIBRARY
24 $      SET NOVERIFY
25 $ System_build:
26 $      @ELN_EBUILD_1    LAB_2B
27 $ Switch_off_verify:
28 $      SET NOVERIFY
29 $      EXIT

```

The .DAT file used by EBUILD looks like this:

```

1      characteristic /noconsole
2      program LAB_2B /debug

```

SOLUTIONS TO EXERCISES

K.9 SOLUTION TO EXERCISE 2 - LAB_2_C.C

```

1  /* -----
2  * SOURCE:          LAB_2_C.C
3  *
4  * PURPOSE:        To calculate temperatures on one of four scales:
5  *                  1) Celsius      (ice melts: 0.0, water boils 100.0)
6  *                  2) Fahrenheit  ( 32.0, 212.0)
7  *                  3) Kelvin      (273.0, 373.0)
8  *                  4) Reaumur     ( 0.0, 80.0)
9  *                  given an input on one of these scales.
10 *                  A subprocess is created to perform the conversion
11 *
12 * COMPILE:        $ CC /LIST /DEBUG LAB_2_C + ELN$:VAXELNC /LIBRARY
13 *
14 * LINK:           $ LINK /DEBUG LAB_2_C, -
15 *                  _$ ELN$:CRTLSHARE /LIBRARY, -
16 *                  _$ ELN$:RTLSHARE /LIBRARY, -
17 *                  _$ ELN$:RTL /LIBRARY
18 *
19 * BUILD:          $ EBUILD /NOEDIT LAB_2_C
20 *
21 * NOTES:          1) Command procedure LAB_2_C.COM compiles, links
22 *                  and builds this module into a VAXELN system or
23 *                  for running under VMS
24 * -----
25 */
26
27 #include $vaxelnc
28 #include ctype      /* for the toupper function */
29
30 BOOLEAN user_continues = TRUE;
31
32 float   temp,
33         temp_value_in,
34         temp_value_out;
35
36 char    answer,
37         temp_scale_in,
38         temp_scale_out;
39
40 PROCESS process_id;
41
42 int     status_returned;
43

```

```

44 lab_2 ()
45     MAIN_PROGRAM
46     {
47         float   fahrenheit_to_celsius(),
48                 kelvin_to_celsius(),
49                 reaumur_to_celsius(),
50                 celsius_to_fahrenheit(),
51                 celsius_to_kelvin(),
52                 celsius_to_reaumur();
53
54         void     temperature_server();
55
56         char     dummy;
57
58     while (user_continues)
59     {
60         printf ("\n\n");
61         printf ("Please enter temperature: ");
62         scanf ("%f", &temp_value_in);
63
64         dummy = getchar();    /* dispose of end-of-line */
65         printf ("\nPlease enter scale for that temperature [K,C,F,R]: ");
66         scanf ("%c", &temp_scale_in);
67
68         /* uppercase temp_scale_in as necessary
69        */
70         temp_scale_in = toupper (temp_scale_in);
71
72         dummy = getchar();    /* dispose of end-of-line */
73         printf ("Please enter the conversion required [K,C,F,R]: ");
74         scanf ("%c", &temp_scale_out);
75
76         /* uppercase temp_scale_out as necessary
77        */
78         temp_scale_out = toupper (temp_scale_out);
79
80         dummy = getchar();    /* dispose of end-of-line */
81
82     ker$create_process (&status_returned,
83                        &process_id,
84                        temperature_server,
85                        NULL);
86         if (! (status_returned & 1))
87             printf ("KER$CREATE_PROCESS status was: %d\n", status_returned);
88

```

SOLUTIONS TO EXERCISES

```

89  ker$wait_any (&status_returned,
90               NULL,
91               NULL,
92               process_id);
93      if (! (status_returned & 1))
94      printf ("KER$WAIT_ANY status was: %d\n", status_returned);
95
96  printf ("Temperature before conversion: %6.1f deg%c\n",
97          temp_value_in, temp_scale_in);
98  printf ("Temperature after conversion: %6.1f deg%c\n",
99          temp_value_out, temp_scale_out);
100
101  printf ("Do you wish to try again? [Y/N]: ");
102  scanf ("%c", &answer);
103
104  user_continues = (answer == 'Y' || answer == 'y');
105
106      } /* while user continues */
107
108  }
109
110  /* ===== Temperature conversion functions =====
111  */
112
113  float fahrenheit_to_celsius (t)
114  float t;
115      {
116      return ((t-32.0) * 5.0 / 9.0);
117      }
118
119  /* ----- */
120
121  float kelvin_to_celsius (t)
122  float t;
123      {
124      return (t - 273.0);
125      }
126
127  /* ----- */
128
129  float reaumur_to_celsius (t)
130  float t;
131      {
132      return (t * 1.25);
133      }
134

```

```
135 /* ----- */
136
137 float celsius_to_fahrenheit (t)
138 float t;
139 {
140     return (t * 1.8 + 32.0);
141 }
142
143 /* ----- */
144
145 float celsius_to_kelvin (t)
146 float t;
147 {
148     return (t + 273.0);
149 }
150
151 /* ----- */
152
153 float celsius_to_reaumur (t)
154 float t;
155 {
156     return (t * 0.8);
157 }
158
```

SOLUTIONS TO EXERCISES

```
159  /* -----
160                                     <Subprocess block>
161  -----
162  */
163
164  void temperature_server()
165  {
166      switch (temp_scale_in )
167      {
168          case 'F': temp = fahrenheit_to_celsius (temp_value_in);
169                     break;
170          case 'R': temp = reamur_to_celsius      (temp_value_in);
171                     break;
172          case 'K': temp = kelvin_to_celsius      (temp_value_in);
173                     break;
174          case 'C': temp = temp_value_in;
175                     break;
176      } /* switch temp_scale_in */
177
178      switch (temp_scale_out)
179      {
180          case 'C': temp_value_out = temp;
181                     break;
182          case 'F': temp_value_out = celsius_to_fahrenheit (temp);
183                     break;
184          case 'R': temp_value_out = celsius_to_reamur (temp);
185                     break;
186          case 'K': temp_value_out = celsius_to_kelvin (temp);
187                     break;
188      } /* switch temp_scale_out */
189
190      return;
191  }
```

K.9.1 Running LAB_2_C

```

1  $      ! LAB_2_C.COM
2  $      ! Command procedure to build the VAXELN module LAB_2_C
3  $      ! into a system
4  $      !
5  $      ON ERROR THEN GOTO Switch_off_verify
6  $      SET DEFAULT Default_directory
7  $      SET VERIFY
8  $ !
9  $ Compile_LAB_2_C:
10 $      CC -
11         /LIST -
12         /DEBUG LAB_2_C + ELN$:VAXELNC /LIBRARY
13 $ Link_LAB_2_C:
14 $      LINK -
15         /DEBUG LAB_2_C -
16         , ELN$:CRTLSHARE /LIBRARY -
17         ,      RTLSHARE /LIBRARY -
18         ,      RTL      /LIBRARY
19 $      SET NOVERIFY
20 $ System_build:
21 $      @ELN_EBUILD_1 LAB_2_C
22 $ Switch_off_verify:
23 $      SET NOVERIFY
24 $      EXIT

```

The .DAT file looks like this:

```

1      characteristic /noconsole
2      program LAB_2_C /debug

```


SOLUTIONS TO EXERCISES

Output from LAB_2_C:

SH SYS

! Available: Pages: 1391, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle: 0 00:00:03.25 Total: 0 00:00:03.39
! Time used by past jobs: 0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program LAB_2_C, priority 16 is waiting.
!

CANCEL CONTROL

GO

! Job 5, process 1, program LAB_2_C running.

!

!Please enter temperature: 32

!Please enter scale for that temperature [K,C,F,R]: f

!Please enter the conversion required [K,C,F,R]: c

!Temperature before conversion: 32.0 degF

!Temperature after conversion: 0.0 degC

!Do you wish to try again? [Y/N]: y

!

!Please enter temperature: 45

!Please enter scale for that temperature [K,C,F,R]: f

!Please enter the conversion required [K,C,F,R]: k

!Temperature before conversion: 45.0 degF

!Temperature after conversion: 280.2 degK

!Do you wish to try again? [Y/N]: y

!

!Please enter temperature: 80

!Please enter scale for that temperature [K,C,F,R]: r

!Please enter the conversion required [K,C,F,R]: k

!Temperature before conversion: 80.0 degR

!Temperature after conversion: 373.0 degK

!Do you wish to try again? [Y/N]: n

!

! Job 5, process 1, program LAB_2_C has exited.

!

exit

K.10 SOLUTION TO EXERCISE 3 - LAB_3.PAS

```

1  {-----
2  SOURCE:          LAB_3.PAS
3
4  PURPOSE:         To calculate temperatures on one of four scales:
5                    1) Celsius      (ice melts: 0.0, water boils 100.0)
6                    2) Fahrenheit   ( 32.0, 212.0)
7                    3) Kelvin       (273.0, 373.0)
8                    4) Reaumur      (  0.0,  80.0)
9                    given an input on one of these scales.
10                   This program creates a job and passes to it, via
11                   arguments, the temperature conversion information.
12                   The new job performs the conversion and prints the
13                   results.
14
15  COMPILE:         $ EPASCAL /LIST /DEBUG LAB_3
16
17  LINK:            $ LINK /DEBUG LAB_3, -
18                   _$ ELN$:RTLSHARE /LIBRARY, -
19                   _$ ELN$:RTL /LIBRARY
20
21  BUILD:           $ EBUILD /NOEDIT LAB_3
22
23  NOTES:           1) Command procedure LAB_3.COM compiles, links
24                   and builds this module into a VAXELN system
25  -----}
26  MODULE Lab_3 [IDENT ('V1.000')];
27
28  CONST
29      LF      = '(10);  {line-feed}
30      Strlen  = 10;
31  VAR
32      User_continues  : BOOLEAN := TRUE;
33
34      Temp,
35      Temp_value_in,
36      Temp_value_out  : REAL;
37
38      Status_returned : INTEGER;
39
40      Answer,
41      Temp_scale_in,
42      Temp_scale_out  : CHAR;
43
44      Temp_in_string  : VARYING_STRING(Strlen);
45

```

SOLUTIONS TO EXERCISES

```

46 {----- PROGRAM BLOCK -----}
47
48 PROGRAM Lab_3 (OUTPUT);
49
50 VAR
51     This_jobs_port,
52     New_jobs_port   : PORT;
53
54     PROCEDURE Ensure_uppercase ( VAR C : CHAR );
55
56     BEGIN
57         IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
58     END {PROCEDURE Ensure_uppercase};
59
60 BEGIN
61
62 WHILE User_continues DO
63
64 BEGIN
65     WRITELN ( LF );
66
67     WRITE ( 'Please enter temperature: ' );
68     READLN ( Temp_value_in );
69
70     WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
71     READLN ( Temp_scale_in );
72     Ensure_uppercase ( Temp_scale_in );
73
74     WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
75     READLN ( Temp_scale_out );
76     Ensure_uppercase ( Temp_scale_out );
77
78 Temp_in_string := CONVERT ( VARYING_STRING(Strlen), Temp_value_in );
79
80 { VAXELN seems to prefer a fresh port for each termination message -
81   re-using the same port for exit messages causes the WAIT_ANY below
82   to complete immediately }
83
84 CREATE_PORT ( This_jobs_port,
85              STATUS := Status_returned );
86
87     IF NOT ODD ( Status_returned ) THEN
88         WRITELN ( 'CREATE_PORT status was: ', Status_returned :1);
89

```

```

90  CREATE_JOB ( New_jobs_port,
91                'LAB_3A',
92                '',
93                '',
94                Temp_in_string,
95                Temp_scale_in,
96                Temp_scale_out,
97                NOTIFY := This_jobs_port,
98                STATUS := Status_returned );
99
100         IF NOT ODD ( Status_returned ) THEN
101         WRITELN ( 'CREATE_JOB status was: ', Status_returned :1);
102
103  WAIT_ANY ( This_jobs_port,
104             STATUS := Status_returned );
105         IF NOT ODD ( Status_returned ) THEN
106         WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);
107
108  { get rid of old PORT object to conserve system space }
109
110  DELETE ( This_jobs_port,
111           STATUS := Status_returned );
112
113         IF NOT ODD ( Status_returned ) THEN
114         WRITELN ( 'DELETE status was: ', Status_returned :1);
115
116  WRITE ( 'Do you wish to try again? [Y/N]: ' );
117  READLN ( Answer );
118
119  User_continues := Answer IN ['Y', 'y'];
120
121  END;
122
123  END {of PROGRAM}.
124  END {of MODULE };

```

SOLUTIONS TO EXERCISES

K.11 SOLUTION TO EXERCISE 3 - LAB_3A.PAS

This is the code run by the job created from LAB_3.

```
1  {-----}
2  SOURCE:      LAB_3A.PAS
3
4  PURPOSE:     This program is run from LAB_3 and does the temperature
5                conversions and displays the results
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG LAB_3A
8
9  LINK:        $ LINK /DEBUG LAB_3A, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ ELN$:RTL /LIBRARY
12
13 BUILD:       $ EBUILD /NOEDIT LAB_3
14
15 NOTES:       1) Command procedure LAB_3.COM compiles, links
16                and builds this module into a VAXELN system
17  -----}
18 MODULE Lab_3a [IDENT ('V1.000')];
19
20 CONST
21     LF      = '(10);    {line-feed}
22
23 VAR
24     Temp,
25     Temp_value_in,
26     Temp_value_out  : REAL;
27
28     Status_returned : INTEGER;
29
30     Temp_scale_in,
31     Temp_scale_out  : CHAR;
32
33     Temp_in_string  : VARYING_STRING(10);
34
```

```

35 {----- FUNCTION DECLARATIONS ----- }
36
37 FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;
38
39 FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;
40
41     BEGIN
42         Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
43     END;
44
45 {----- }
46
47 FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;
48
49     BEGIN
50         Kelvin_to_Celsius := T - 273.0;
51     END;
52
53 {----- }
54
55 FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;
56
57     BEGIN
58         Reaumur_to_Celsius := T * 1.25;
59     END;
60
61 {----- }
62
63 FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;
64
65     BEGIN
66         Celsius_to_Fahrenheit := T * 1.8 + 32.0;
67     END;
68
69 {----- }
70
71 FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;
72
73     BEGIN
74         Celsius_to_Kelvin := T + 273.0;
75     END;
76
77 {----- }
78
79 FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;
80
81     BEGIN
82         Celsius_to_Reaumur := T * 0.8;
83     END;
84
85 {----- END FUNCTION DECLARATIONS ----- }
86

```

SOLUTIONS TO EXERCISES

```

87  {----- PROGRAM BLOCK -----}
88
89  PROGRAM Lab_3a (OUTPUT);
90
91  CONST
92      LF = ''(10);    { line-feed }
93
94  BEGIN
95
96  Temp_in_string := PROGRAM_ARGUMENT ( 3 );
97  Temp_scale_in  := PROGRAM_ARGUMENT ( 4 );
98  Temp_scale_out := PROGRAM_ARGUMENT ( 5 );
99
100 Temp_value_in  := CONVERT ( REAL, Temp_in_string );
101
102 CASE Temp_scale_in OF
103     'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
104
105     'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );
106
107     'K' : Temp := Kelvin_to_Celsius    ( Temp_value_in );
108
109     'C' : Temp := Temp_value_in;
110 END; {CASE Temp_scale_in}
111
112 { provide conversion }
113
114 CASE Temp_scale_out OF
115     'C' : Temp_value_out := Temp;
116
117     'F' : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
118
119     'R' : Temp_value_out := Celsius_to_Reaumur    ( Temp );
120
121     'K' : Temp_value_out := Celsius_to_Kelvin    ( Temp );
122 END; {CASE temp_scale_out}
123
124 WRITELN ( LF, LF );
125
126 WRITELN ( 'Temperature before conversion: ',
127           Temp_value_in :6:1,
128           ' deg',
129           Temp_scale_in );
130
131 WRITELN ( 'Temperature after conversion: ',
132           Temp_value_out :6:1,
133           ' deg',
134           Temp_scale_out );
135
136 END {of PROGRAM}.
137 END {of MODULE };

```

K.11.1 Running LAB_3

```

1 $      ! LAB_3.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! modules LAB_3 and LAB_3A
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $ Compile:
9 $      @ELN_COMPILE_1 LAB_3
10 $     @ELN_COMPILE_1 LAB_3A
11 $ Link:
12 $     @ELN_LINK_1     LAB_3
13 $     @ELN_LINK_1     LAB_3A
14 $ System_build:
15 $     @ELN_EBUILD_1   LAB_3
16 $ Switch_off_verify:
17 $     SET NOVERIFY
18 $     EXIT

```

The .DAT file used by EBUILD looks like this:

```

1     characteristic /noconsole /nonetwork /nofile /noserver
2     program LAB_3 /debug
3     program LAB_3A /norun /debug

```


SOLUTIONS TO EXERCISES

Output from LAB_3:

```
SH SYS
! Available: Pages: 17885, Page table slots: 53, Pool blocks: 283
! Time since SET TIME: Idle:  0 00:00:59.77 Total:  0 00:00:59.93
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program LAB_3, priority 16 is waiting.
!
GO
! Job 4, process 1, program LAB_3 running.
!
!Please enter temperature: 32.0
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: C
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_3A.EXE;1
!
! Job 5, process 1, program LAB_3A needs attention.
! Module LAB_3A
! 93:
!>>94: BEGIN
! 95:
! 96: Temp_in_string := PROGRAM_ARGUMENT ( 3 );
! 97: Temp_scale_in  := PROGRAM_ARGUMENT ( 4 );
! 98: Temp_scale_out := PROGRAM_ARGUMENT ( 5 );
GO
! Job 5, process 1, program LAB_3A running.
!
!Temperature before conversion:  32.0 degF
!Temperature after conversion:    0.0 degC
! Job 5, process 1, program LAB_3A has exited.
!
! Job 4, process 1, program LAB_3 running.
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 30.5
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: K
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_3A.EXE;1
!
```

```

! Job 6, process 1, program LAB_3A needs attention.
!%DEBUG-W-OPENIN, error opening as input
! Module name      Routine or Psect name      Line      Rel PC      Abs PC
! LAB_3A           LAB_3A                     94         00000002  000004DD
! %Line 94 + 0000: MOVAB -6C(SP),SP
GO
! Job 6, process 1, program LAB_3A running.
!
!Temperature before conversion:  30.5 degC
!Temperature after conversion:   303.5 degK
! Job 6, process 1, program LAB_3A has exited.
!
! Job 4, process 1, program LAB_3 running.
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: -23.4
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: R
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_3A.EXE;1
!
! Job 7, process 1, program LAB_3A needs attention.
!%DEBUG-W-OPENIN, error opening as input
! Module name      Routine or Psect name      Line      Rel PC      Abs PC
! LAB_3A           LAB_3A                     94         00000002  000004DD
! %Line 94 + 0000: MOVAB -6C(SP),SP
GO
! Job 7, process 1, program LAB_3A running.
!
!Temperature before conversion:  -23.4 degF
!Temperature after conversion:   -24.6 degR
! Job 7, process 1, program LAB_3A has exited.
!
! Job 4, process 1, program LAB_3 running.
!Do you wish to try again? [Y/N]: N
! Job 4, process 1, program LAB_3 has exited.
!
EXIT

```

The warning messages from EDEBUG are probably a result of loading the same program more than once and only specifying the image once in the .DAT file to EBUILD.

SOLUTIONS TO EXERCISES

K.12 SOLUTION TO EXERCISE 4 - LAB_4.PAS

```

1  {-----}
2  SOURCE:      LAB_4.PAS
3
4  PURPOSE:     To demonstrate creation and access to a VAXELN AREA
5
6  COMPILE:     $ EPASCAL /LIST /DEBUG LAB_4, -
7               _$ VAXELN-MODULES /LIBRARY
8
9  LINK:        $ LINK /DEBUG LAB_4, VAXELN-MODULES /LIBRARY, -
10              _$ ELN$:RTLSHARE /LIBRARY, -
11              _$ ELN$:RTL /LIBRARY /INCLUDE= -
12              _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
13              _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
14
15 BUILD:       $ EBUILD /NOEDIT LAB_4
16
17 NOTES:       1) Command procedure LAB_4.COM compiles, links and
18              builds this module into a VAXELN system
19  {-----}
20 MODULE Lab_4 [IDENT ('V1.000')];
21
22 INCLUDE
23     Check_status_and_report;
24
25 CONST
26     Limit                = 100;
27     Name_of_area         = 'Celsius_table';
28
29 { define the area with its preamble }
30 TYPE
31     Temperature_table = RECORD
32         First_value  : INTEGER;
33         Last_value   : INTEGER;
34         Scale_name   : CHAR;
35         Scale_wanted : CHAR;
36         Temperatures : ARRAY [1..Limit] OF REAL;
37     END;
38
39 VAR
40     Area_identity      : AREA;
41     This_port,
42     New_jobs_port      : PORT;
43     Start_of_area      : ^Temperature_table;
44     Returned_status    : INTEGER;
45
46 {-----}
47

```

```

48 PROGRAM Lab_4;
49
50 VAR
51     I                : 1..Limit;
52
53 BEGIN
54
55     CREATE_AREA ( Area_identity,
56                   Start_of_area,
57                   Name_of_area,
58                   STATUS := Returned_status );
59
60     Check_status_and_report ( Returned_status,
61                               'CREATE_AREA' );
62
63     WITH Start_of_area^ DO
64     BEGIN
65         First_value := 1;
66         Last_value  := First_value + Limit - 1;
67         Scale_name   := 'C';
68         Scale_wanted := 'F';
69
70     FOR I := First_value TO Last_value DO Temperatures[I] := I * 1.0;
71 END;
72
73
74     CREATE_PORT ( This_port,
75                  STATUS := Returned_status );
76
77     Check_status_and_report ( Returned_status,
78                               'CREATE_PORT' );
79
80     CREATE_JOB ( New_jobs_port,
81                 'LAB_4A',
82                 '',
83                 '',
84                 Name_of_area,
85                 NOTIFY := This_port,
86                 STATUS := Returned_status );
87
88     Check_status_and_report ( Returned_status,
89                               'CREATE_JOB' );
90
91     WAIT_ANY ( This_port,
92               STATUS := returned_status );
93
94     Check_status_and_report ( Returned_status,
95                               'WAIT_ANY' );
96
97 END { of PROGRAM}.
98 END { of MODULE };

```

SOLUTIONS TO EXERCISES

K.13 SOLUTION TO EXERCISE 4 - LAB_4A.PAS

This is the code run by the job created from LAB_4.

```

1  {-----}
2  SOURCE:          LAB_4A.PAS
3
4  PURPOSE:         This program is run from LAB_4 and does the temperature
5                    conversions and displays the results
6
7  COMPILE:         $ EPASCAL /LIST /DEBUG LAB_4A
8
9  LINK:            $ LINK /DEBUG LAB_4A, -
10                  _$ ELN$:RTLSHARE /LIBRARY, -
11                  _$ ELN$:RTL /LIBRARY
12
13 BUILD:           $ EBUILD /NOEDIT LAB_4
14
15 NOTES:           1) Command procedure LAB_4.COM compiles, links
16                  and builds this module into a VAXELN system
17 -----}
18 MODULE Lab_4a [IDENT ('V1.000')];
19
20 CONST
21     LF          = '(10);    {line-feed}
22     Limit       = 100;
23
24 TYPE
25     Temperature_table = RECORD
26         First_value   : INTEGER;
27         Last_value    : INTEGER;
28         Scale_name     : CHAR;
29         Scale_wanted   : CHAR;
30         Temperatures   : ARRAY [1..Limit] OF REAL;
31     END;
32
33 VAR
34     Area_ID          : AREA;
35     This_port,
36     New_jobs_port    : PORT;
37     Name_of_area     : VARYING_STRING(31);
38     Start_of_area    : ^Temperature_table;
39
40     Returned_status   : INTEGER;
41
42 { Temperature conversion function definitions here... }
43
44     %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'
45
46 {----- PROGRAM BLOCK -----}
47

```

```

48 PROGRAM Lab_4a (OUTPUT);
49
50 VAR
51     I : INTEGER;
52
53 BEGIN
54
55     Name_of_area      := PROGRAM_ARGUMENT ( 3 );
56
57     CREATE_AREA (     Area_ID,
58                     Start_of_area,
59                     Name_of_area,
60                     STATUS := Returned_status );
61     IF NOT ODD ( Returned_status ) THEN
62         WRITELN ( 'CREATE_AREA status was: ', Returned_status :1 );
63
64     WITH Start_of_area^ DO
65
66     BEGIN
67
68     CASE Scale_name OF
69
70         'F', 'f' : FOR I := First_value TO Last_value DO
71                     Temperatures[I] :=
72                         Fahrenheit_to_Celsius ( Temperatures[I] );
73
74         'R', 'r' : FOR I := First_value TO Last_value DO
75                     Temperatures[I] :=
76                         Reaumur_to_Celsius ( Temperatures[I] );
77
78         'K', 'k' : FOR I := First_value TO Last_value DO
79                     Temperatures[I] :=
80                         Kelvin_to_Celsius ( Temperatures[I] );
81
82     END; {CASE Scale_name}
83
84     CASE Scale_wanted OF
85
86         'F', 'f' : FOR I := First_value TO Last_value DO
87                     Temperatures[I] :=
88                         Celsius_to_Fahrenheit ( Temperatures[I] );
89
90         'R', 'r' : FOR I := First_value TO Last_value DO
91                     Temperatures[I] :=
92                         Celsius_to_Reaumur ( Temperatures[I] );
93
94         'K', 'k' : FOR I := First_value TO Last_value DO
95                     Temperatures[I] :=
96                         Celsius_to_Kelvin ( Temperatures[I] );
97
98     END; {CASE Scale_wanted}

```

SOLUTIONS TO EXERCISES

```

99
100 END; {WITH}
101
102 CREATE_PORT ( This_port,
103               STATUS := Returned_status );
104
105         IF NOT ODD ( Returned_status ) THEN
106             WRITELN ( 'CREATE_PORT status was: ', Returned_status :1 );
107
108 CREATE_JOB   ( New_jobs_port,
109               'LAB_4B',
110               '',
111               '',
112               Name_of_area,
113               NOTIFY := This_port,
114               STATUS  := Returned_status );
115
116         IF NOT ODD ( Returned_status ) THEN
117             WRITELN ( 'CREATE_JOB status was: ', Returned_status :1 );
118
119 WAIT_ANY ( This_port,
120           STATUS := returned_status );
121
122         IF NOT ODD ( Returned_status ) THEN
123             WRITELN ( 'WAIT_ANY status was: ', Returned_status :1 );
124
125 END {of PROGRAM}.
126 END {of MODULE };
```

The file included in the compilation of LAB_4A looks like this:

```

1  {----- FUNCTION DECLARATIONS ----- }
2
3  FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;
4
5  FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;
6
7      BEGIN
8          Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
9      END;
10
11 {----- }
12
13 FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;
14
15     BEGIN
16         Kelvin_to_Celsius := T - 273.0;
17     END;
18
19 {----- }
20
21 FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;
22
23     BEGIN
24         Reaumur_to_Celsius := T * 1.25;
25     END;
26
27 {----- }
28
29 FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;
30
31     BEGIN
32         Celsius_to_Fahrenheit := T * 1.8 + 32.0;
33     END;
34
35 {----- }
36
37 FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;
38
39     BEGIN
40         Celsius_to_Kelvin := T + 273.0;
41     END;
42
43 {----- }
44
45 FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;
46
47     BEGIN
48         Celsius_to_Reaumur := T * 0.8;
49     END;
50
51 {----- END FUNCTION DECLARATIONS ----- }

```


SOLUTIONS TO EXERCISES

K.14 SOLUTION TO EXERCISE 4 - LAB_4B.PAS

This is the code run by the job created from LAB_4A.

```

1  {-----}
2  SOURCE:      LAB_4B.PAS
3
4  PURPOSE:      This program is run from LAB_4A and displays the results
5                  of the temperature conversion
6
7  COMPILE:      $ EPASCAL /LIST /DEBUG LAB_4B
8
9  LINK:         $ LINK /DEBUG LAB_4B, -
10                _$ ELN$:RTLSHARE /LIBRARY, -
11                _$ ELN$:RTL /LIBRARY
12
13 BUILD:        $ EBUILD /NOEDIT LAB_4
14
15 NOTES:        1) Command procedure LAB_4.COM compiles, links
16                  and builds this module into a VAXELN system
17 -----}
18 MODULE Lab_4b [IDENT ('V1.000')];
19
20 CONST
21     LF      = '(10);    {line-feed}
22     Limit   = 100;
23
24 TYPE
25     Temperature_table = RECORD
26         First_value   : INTEGER;
27         Last_value    : INTEGER;
28         Scale_name     : CHAR;
29         Scale_wanted   : CHAR;
30         Temperatures  : ARRAY [1..Limit] OF REAL;
31     END;
32
33 VAR
34     Area_ID           : AREA;
35     Name_of_area      : VARYING_STRING(31);
36     Start_of_area     : ^Temperature_table;
37     Full_scale_name   : VARYING_STRING(10); {size of Fahrenheit}
38     Returned_status   : INTEGER;
39
40 {----- PROGRAM BLOCK -----}
41
42 PROGRAM Lab_4b (OUTPUT);
43
44 VAR
45     I : INTEGER;
46

```

```

47 BEGIN
48
49 Name_of_area      := PROGRAM_ARGUMENT ( 3 );
50
51 CREATE_AREA (      Area_ID,
52                  Start_of_area,
53                  Name_of_area,
54                  STATUS := Returned_status );
55      IF NOT ODD ( Returned_status ) THEN
56          WRITELN ( 'CREATE_AREA status was: ', Returned_status :1 );
57
58 WITH Start_of_area^ DO
59 BEGIN
60     CASE Scale_wanted OF
61         'C', 'c'      : Full_scale_name := 'Celsius';
62         'F', 'f'      : Full_scale_name := 'Fahrenheit';
63         'K', 'k'      : Full_scale_name := 'Kelvin';
64         'R', 'r'      : Full_scale_name := 'Reaumur';
65
66     END; {CASE}
67
68     WRITELN ( 'Table of temperatures in ', Full_scale_name );
69     WRITELN ( 'From original values ', First_value:1,
70               ' to ', Last_value:1, ' in degrees ', Scale_name, LF );
71
72     FOR I := First_value TO Last_value DO
73     BEGIN
74         IF ( (I-1) MOD 10 ) = 0 THEN WRITELN;
75         WRITE ( Temperatures[I]:7:1 );
76     END;
77
78 END {WITH Start_of_area};
79
80 END {of PROGRAM}.
81
82 END {of MODULE };

```

SOLUTIONS TO EXERCISES

K.14.1 Running LAB_4

```
1 $      ! LAB_4.COM
2 $      ! Command procedure to build the VAXELN
3 $      ! modules LAB_4, LAB_4A and LAB_4B
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $      EPASCAL -
10         /DEBUG -
11         /LIST LAB_4, VAXELN-MODULES /LIBRARY
12 $      LINK -
13         /DEBUG LAB_4, VAXELN-MODULES /LIBRARY -
14         ,ELN$:RTLSHARE /LIBRARY -
15         ,RTL /LIBRARY
16 $      SET NOVERIFY
17 $ !
18 $      @ELN_COMPILE_1 LAB_4A
19 $      @ELN_COMPILE_1 LAB_4B
20 $      @ELN_LINK_1 LAB_4A
21 $      @ELN_LINK_1 LAB_4B
22 $ System_build:
23 $      @ELN_EBUILD_1 LAB_4
24 $ Switch_off_verify:
25 $      SET NOVERIFY
26 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole /nofile
2      program LAB_4 /debug
3      program LAB_4A /norun /debug
4      program LAB_4B /norun /debug
```

Output from LAB_4:

```

SH SYS
! Available: Pages: 17877, Page table slots: 53, Pool blocks: 280
! Time since SET TIME: Idle:  0 00:00:05.84 Total:  0 00:00:06.00
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program LAB_4, priority 16 is waiting.
!
GO
! Job 4, process 1, program LAB_4 running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_4A.EXE;1
!
! Job 5, process 1, program LAB_4A needs attention.
! Module LAB_4A
! 103:
!>>104: BEGIN
! 105:
! 106: Name_of_area      := PROGRAM_ARGUMENT ( 3 );
! 107:
! 108: CREATE_AREA (    Area_ID,
GO
! Job 5, process 1, program LAB_4A running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_4B.EXE;1
!
! Job 6, process 1, program LAB_4B needs attention.
! Module LAB_4B
! 46:
!>>47: BEGIN
! 48:
! 49: Name_of_area      := PROGRAM_ARGUMENT ( 3 );
! 50:
! 51: CREATE_AREA (    Area_ID,
GO

```

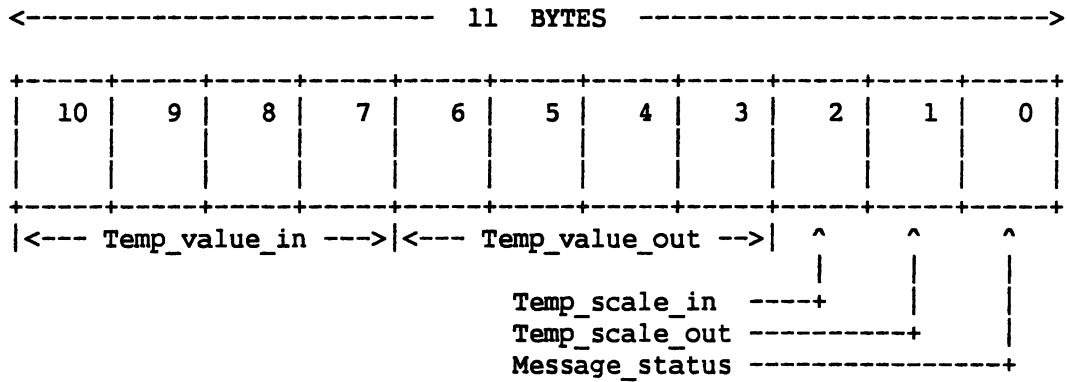
SOLUTIONS TO EXERCISES

```
! Job 6, process 1, program LAB_4B running.
!Table of temperatures in Fahrenheit
!From original values 1 to 100 in degrees C
!
!  33.8  35.6  37.4  39.2  41.0  42.8  44.6  46.4  48.2  50.0
!  51.8  53.6  55.4  57.2  59.0  60.8  62.6  64.4  66.2  68.0
!  69.8  71.6  73.4  75.2  77.0  78.8  80.6  82.4  84.2  86.0
!  87.8  89.6  91.4  93.2  95.0  96.8  98.6 100.4 102.2 104.0
! 105.8 107.6 109.4 111.2 113.0 114.8 116.6 118.4 120.2 122.0
! 123.8 125.6 127.4 129.2 131.0 132.8 134.6 136.4 138.2 140.0
! 141.8 143.6 145.4 147.2 149.0 150.8 152.6 154.4 156.2 158.0
! 159.8 161.6 163.4 165.2 167.0 168.8 170.6 172.4 174.2 176.0
! 177.8 179.6 181.4 183.2 185.0 186.8 188.6 190.4 192.2 194.0
! 195.8 197.6 199.4 201.2 203.0 204.8 206.6 208.4 210.2 212.0
! Job 6, process 1, program LAB_4B has exited.
!
! Job 4, process 1, program LAB_4 running.
! Job 5, process 1, program LAB_4A has exited.
!
! Job 4, process 1, program LAB_4 running.
! Job 4, process 1, program LAB_4 has exited.
!
EXIT
```

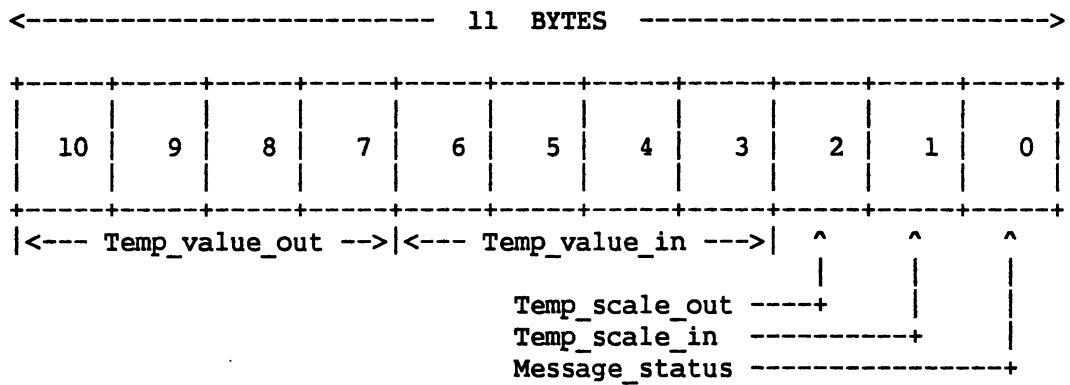
K.15 SOLUTION TO EXERCISE 5 - LAB_5.PAS

These solutions use a message packet. The packet contents are defined slightly differently for each program as shown below:

Packet as defined for LAB_5:



Packet as defined for LAB_5A:



SOLUTIONS TO EXERCISES

```

1  {-----}
2  SOURCE:      LAB_5.PAS
3
4  PURPOSE:      To demonstrate creation of a circuit between VAXELN jobs
5                  and two-way communication through it.
6                  This program prompts for temperature information
7                  then passes a request to LAB_5A for conversion and
8                  return of the results via the circuit established
9                  by LAB_5A
10
11  COMPILE:      $ EPASCAL /LIST /DEBUG LAB_5, -
12                  _$ VAXELN-MODULES /LIBRARY
13
14  LINK:         $ LINK /DEBUG LAB_5, VAXELN-MODULES /LIBRARY, -
15                  _$ ELN$:RTLSHARE /LIBRARY, -
16                  _$ ELN$:RTL /LIBRARY /INCLUDE= -
17                  _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
18                  _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
19
20  BUILD:        $ EBUILD /NOEDIT LAB_5
21
22  NOTES:        1) Command procedure LAB_5.COM compiles, links
23                  and builds this module into a VAXELN system
24  -----}
25  MODULE Lab_5 [IDENT ('V1.000')];
26
27  INCLUDE
28      Check_status_and_report;
29
30  CONST
31      LF      = '(10);      {line-feed}
32  TYPE
33      Message_kind      = ( Continue, Finished );
34
35      Message_type      = RECORD
36          Message_status : Message_kind;
37          Temp_scale_out,
38          Temp_scale_in  : CHAR;
39          Temp_value_out,
40          Temp_value_in  : REAL;
41      END;
42

```

```

43 {----- PROGRAM BLOCK -----}
44
45 PROGRAM Lab_5 (OUTPUT);
46
47 VAR
48     Answer           : CHAR;
49     User_continues    : BOOLEAN := TRUE;
50     My_new_port       : PORT;
51     Outgoing_message,
52     Incoming_message  : ^Message_type;
53     Message_identity  : MESSAGE;
54     My_new_port_name  : NAME;
55     Returned_status    : INTEGER;
56     Temperature_sent  : REAL;
57     Scale_sent        : CHAR;
58
59     PROCEDURE Ensure_uppercase ( VAR C : CHAR );
60
61     BEGIN
62         IF C IN ['a'..'z'] THEN C := CHR ( ORD ( C ) - 32 );
63     END {PROCEDURE Ensure_uppercase};
64
65 { MAIN program start }
66
67 BEGIN
68
69     CREATE_PORT (    My_new_port,
70                     STATUS := Returned_status );
71
72     Check_status_and_report ( Returned_status,
73                               'CREATE_PORT' );
74
75     CREATE_NAME (    My_new_port_name,
76                     'MASTER_PORT',
77                     My_new_port,
78                     TABLE := NAME$UNIVERSAL,
79                     STATUS := Returned_status );
80
81     Check_status_and_report ( Returned_status,
82                               'CREATE_NAME' );
83
84     INITIALIZATION_DONE ( STATUS := Returned_status );
85
86     Check_status_and_report ( Returned_status,
87                               'INITIALIZATION_DONE' );
88
89     ACCEPT_CIRCUIT ( My_new_port,
90                     STATUS := Returned_status );
91
92     Check_status_and_report ( Returned_status,
93                               'ACCEPT_CIRCUIT' );

```


SOLUTIONS TO EXERCISES

```

94
95 WHILE User_continues DO
96
97 BEGIN
98
99 CREATE_MESSAGE ( Message_identity,
100                  Outgoing_message,
101                  STATUS := Returned_status );
102
103          Check_status_and_report ( Returned_status,
104                                   'CREATE_MESSAGE' );
105
106 { Initialize the message packet and put 'continue' flag in status.
107   Preserve outgoing temp and scale values as they are 'lost' from
108   outgoing message packet once sent (message pointers are undefined
109   after a SEND) }
110
111 WITH Outgoing_message^ DO
112
113 BEGIN
114   Message_status := Continue;
115
116   WRITELN ( LF );
117   WRITE ( 'Please enter temperature: ' );
118   READLN ( Temp_value_out );
119   Temperature_sent := Temp_value_out;
120
121   WRITE ( 'Please enter scale for that temperature [K,C,F,R]: ' );
122   READLN ( Temp_scale_out );
123   Ensure_uppercase ( Temp_scale_out );
124   Scale_sent := Temp_scale_out;
125
126   WRITE ( 'Please enter the conversion required [K,C,F,R]: ' );
127   READLN ( Temp_scale_in );
128   Ensure_uppercase ( Temp_scale_in );
129   END;
130
131 SEND          ( Message_identity,
132                My_new_port,
133                STATUS := Returned_status );
134
135          Check_status_and_report ( Returned_status,
136                                   'SEND' );
137

```

```

138 WAIT_ANY ( My_new_port );
139
140 RECEIVE ( Message_identity,
141           Incoming_message,
142           My_new_port,
143           STATUS := Returned_status );
144
145           Check_status_and_report ( Returned_status,
146                                   'RECEIVE' );
147
148 WRITELN ( 'Temperature before conversion: ',
149           Temperature_sent :6:1,
150           ' deg',
151           Scale_sent );
152
153 WITH Incoming_message^ DO
154     WRITELN ( 'Temperature after conversion: ',
155             Temp_value_in :6:1,
156             ' deg',
157             Temp_scale_in );
158
159     WRITE ( 'Do you wish to try again? [Y/N]: ' );
160     READLN ( Answer );
161
162     User_continues := Answer IN ['Y', 'y'];
163
164 END;
165
166 CREATE_MESSAGE ( Message_identity,
167                 Outgoing_message,
168                 STATUS := Returned_status );
169
170           Check_status_and_report ( Returned_status,
171                                   'CREATE_MESSAGE' );
172
173 { now that the user has finished put 'finished flag in status }
174
175 WITH Outgoing_message^ DO
176     Message_status := Finished;
177
178 SEND ( Message_identity,
179        My_new_port,
180        STATUS := Returned_status );
181
182           Check_status_and_report ( Returned_status,
183                                   'SEND' );
184
185 END {of PROGRAM}.
186 END {of MODULE };

```

SOLUTIONS TO EXERCISES

K.16 SOLUTION TO EXERCISE 5 - LAB_5A.PAS

This program connects the circuit with LAB_5

```

1  {-----}
2  SOURCE:      LAB_5A.PAS
3
4  PURPOSE:     To demonstrate circuit connection with another job
5                This program connects a circuit with LAB_5 and
6                through that circuit receives conversion requests.
7                After servicing these requests the results are returned
8                on the same circuit.
9
10 COMPILE:     $ EPASCAL /LIST /DEBUG LAB_5A, -
11              _$ VAXELN-MODULES /LIBRARY
12
13 LINK:        $ LINK /DEBUG LAB_5A, VAXELN-MODULES /LIBRARY, -
14              _$ ELN$:RTLSHARE /LIBRARY, -
15              _$ ELN$:RTL /LIBRARY /INCLUDE= -
16              _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
17              _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
18
19 BUILD:       $ EBUILD /NOEDIT LAB_5
20
21 NOTES:       1) Command procedure LAB_5.COM compiles, links
22              and builds this module into a VAXELN system
23 -----}
24 MODULE Lab_5a [IDENT ('V1.000')];
25
26 INCLUDE
27     Check_status_and_report;
28
29 TYPE
30     Message_kind      = ( Continue, Finished );
31
32     Message_type      = RECORD
33         Message_status : Message_kind;
34         Temp_scale_in,
35         Temp_scale_out : CHAR;
36         Temp_value_in,
37         Temp_value_out : REAL;
38     END;
39

```

```

40 {----- PROGRAM BLOCK -----}
41
42 PROGRAM Lab_5a (OUTPUT);
43
44 VAR
45     Process_has_exited      : BOOLEAN := FALSE;
46     Count_of_arguments,
47     Returned_status         : INTEGER;
48     Temp                    : REAL;
49     Incoming_message,
50     Outgoing_message        : ^Message_type;
51     Message_identity         : MESSAGE;
52     New_port                 : PORT;
53
54 { include function definitions }
55
56     %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'
57
58 { MAIN program start }
59
60 BEGIN
61
62 CREATE_PORT      (   New_port,
63                     STATUS := Returned_status );
64
65     Check_status_and_report ( Returned_status,
66                               'CREATE_PORT' );
67
68 CONNECT_CIRCUIT (   New_port,
69                     DESTINATION_NAME := 'MASTER_PORT',
70                     STATUS := Returned_status );
71
72     Check_status_and_report ( Returned_status,
73                               'CONNECT_CIRCUIT' );
74
75 REPEAT
76
77     WAIT_ANY ( New_port );
78
79     RECEIVE (   Message_identity,
80               Incoming_message,
81               New_port,
82               STATUS := Returned_status );
83
84     Check_status_and_report ( Returned_status,
85                               'RECEIVE' );
86

```

SOLUTIONS TO EXERCISES

```

87 { NOTE: Process exits here when message contains FINISHED flag }
88
89 IF Incoming_message^.Message_status = Finished THEN EXIT;
90
91 WITH Incoming_message^ DO
92     BEGIN
93         CASE Temp_scale_in OF
94             'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
95             'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );
96             'K' : Temp := Kelvin_to_Celsius    ( Temp_value_in );
97             'C' : Temp := Temp_value_in;
98         END; {CASE Temp_scale_in}
99     END;
100
101 CREATE_MESSAGE      (  Message_identity,
102                       Outgoing_message,
103                       STATUS := Returned_status );
104
105     Check_status_and_report ( Returned_status,
106                               'CREATE_MESSAGE' );
107
108 WITH Outgoing_message^ DO
109     BEGIN
110         CASE Incoming_message^.Temp_scale_out OF
111             'C' : BEGIN
112                     Temp_value_out := Temp;
113                     Temp_scale_out := 'C';
114                 END;
115             'F' : BEGIN
116                     Temp_value_out := Celsius_to_Fahrenheit ( Temp );
117                     temp_scale_out := 'F';
118                 END;
119             'R' : BEGIN
120                     Temp_value_out := Celsius_to_Reaumur    ( Temp );
121                     Temp_scale_out := 'R';
122                 END;
123             'K' : BEGIN
124                     Temp_value_out := Celsius_to_Kelvin     ( Temp );
125                     Temp_scale_out := 'K';
126                 END;
127         END; {CASE temp_scale_out}
128     END;
129
130
131
132

```

```
133 SEND          (  Message_identity,
134                  New_port,
135                  STATUS := Returned_status );
136
137          Check_status_and_report ( Returned_status,
138                                    'SEND' );
139
140 UNTIL Process_has_exited; { Never satisfies this statement }
141
142 END {of PROGRAM}.
143 END {of MODULE };
```

SOLUTIONS TO EXERCISES

K.16.1 Running LAB_5

```
1 $      ! LAB_5.COM
2 $      ! Command procedure to build the VAXELN modules
3 $      ! LAB_5 and LAB_5A into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $      !
8 $      SET VERIFY
9 $      Compile1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG LAB_5, VAXELN-MODULES /LIBRARY
13 $      Compile2:
14 $      EPASCAL -
15          /LIST -
16          /DEBUG LAB_5A, VAXELN-MODULES /LIBRARY
17 $      Link1:
18 $      LINK -
19          /DEBUG LAB_5 -
20          ,VAXELN-MODULES /LIBRARY -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY /INCLUDE= -
23          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
24          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
25 $      Link2:
26 $      LINK -
27          /DEBUG LAB_5A -
28          ,VAXELN-MODULES /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY /INCLUDE= -
31          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
32          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
33 $      System_build:
34 $      EBUILD -
35          /NOEDIT LAB_5
36 $      Switch_off_verify:
37 $      SET NOVERIFY
38 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_5 /initialize /debug
3      program LAB_5A /debug
```

Output from LAB_5:

```

GO
! Job 4, process 1, program LAB_5 running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_5A.EXE;5
!
! Job 6, process 1, program LAB_5A needs attention.
! Module LAB_5A
! 110:
!>>111: BEGIN
! 112:
! 113: CREATE_PORT      (    New_port,
! 114:                  STATUS := Returned_status );
! 115:
GO
! Job 6, process 1, program LAB_5A running.
!
!Please enter temperature: 32.0
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: K
!Temperature before conversion: 32.0 degF
!Temperature after conversion: 273.0 degK
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 45
!Please enter scale for that temperature [K,C,F,R]: C
!Please enter the conversion required [K,C,F,R]: F
!Temperature before conversion: 45.0 degC
!Temperature after conversion: 113.0 degF
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 273
!Please enter scale for that temperature [K,C,F,R]: K
!Please enter the conversion required [K,C,F,R]: C
!Temperature before conversion: 273.0 degK
!Temperature after conversion: 0.0 degC
!Do you wish to try again? [Y/N]: Y
!
!Please enter temperature: 32.0
!Please enter scale for that temperature [K,C,F,R]: F
!Please enter the conversion required [K,C,F,R]: K
!Temperature before conversion: 32.0 degF
!Temperature after conversion: 273.0 degK
!Do you wish to try again? [Y/N]: N
! Job 6, process 1, program LAB_5A has exited.
!
! Job 4, process 1, program LAB_5 running.
! Job 4, process 1, program LAB_5 has exited.
!
EXIT

```


SOLUTIONS TO EXERCISES

K.17 SOLUTION TO EXERCISE 6 - LAB_6.PAS

```

1  {-----
2  SOURCE:          LAB_6.PAS
3
4  PURPOSE:         This program reads a file of temperatures from a VAX
5                    host into an AREA.
6                    The program then hands over to LAB_6A which converts
7                    those temperatures to a chosen scale and outputs the
8                    results.
9
10 COMPILE:         $ EPASCAL /LIST /DEBUG LAB_6, -
11                  _$ VAXELN-MODULES /LIBRARY
12
13 LINK:            $ LINK /DEBUG LAB_6, VAXELN-MODULES /LIBRARY, -
14                  _$ ELN$:RTLSHARE /LIBRARY, -
15                  _$ ELN$:RTL /LIBRARY /INCLUDE= -
16                  _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
17                  _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
18
19 BUILD:          $ EBUILD /NOEDIT LAB_6
20
21 NOTES:          1) Command procedure LAB_6.COM compiles, links
22                  and builds this module into a VAXELN system
23                  2) Change node and file spec as appropriate
24 -----}
25 MODULE Lab_6 [IDENT ('V1.000')];
26
27 INCLUDE
28     Check_status_and_report;
29
30 CONST
31     First_year      = 1659;
32     Last_year       = 2058;
33
34 TYPE
35     Months          = (January,   February, March,   April,
36                       May,        June,    July,     August,
37                       September, October, November, December,
38                       Annual);
39

```

```

40     Temperature_scale = (Celsius, Fahrenheit, Kelvin, Reaumur);
41
42     Years_record = RECORD
43         Year_number      : First_year..Last_year;
44         Monthly_temps    : ARRAY [January..Annual] OF INTEGER;
45     END;
46
47     Temperature_table = RECORD
48         Latest_year      : First_year..Last_year;
49         Scale_name       : Temperature_scale;
50         Scale_wanted     : Temperature_scale;
51         Temperatures     : ARRAY [First_year..Last_year] OF Years_record;
52     END;
53
54     VAR
55         Temperature_area: AREA;
56         Year             : First_year..Last_year;
57         Start_of_temps   : ^Temperature_table;
58         Cet              : TEXT;
59         This_port,
60         LAB_6As_port     : PORT;
61         Returned_status  : INTEGER;
62
63
64     PROGRAM Lab_6 (INPUT, OUTPUT);
65
66     VAR
67         I                : First_year..Last_year;
68         Period           : Months;
69
70     BEGIN
71
72     { this open statement won't work unless you are on node 1.241 and
73       the device and directory spec are correct }
74
75     OPEN ( Cet,
76         FILE_NAME := '1.241::DISK$COURSEDSK:[SHONE]TEMPS.DAT',
77         HISTORY   := HISTORY$OLD);
78
79     RESET (Cet);
80
81     CREATE_AREA ( Temperature_area,
82                 Start_of_temps,
83                 'ENGLISH_TEMPERATURES',
84                 STATUS := Returned_status );
85
86     Check_status_and_report ( Returned_status,
87                             'CREATE_AREA' );
88

```

SOLUTIONS TO EXERCISES

```

89  I := First_year;
90
91  WHILE NOT EOF(CET) DO
92  BEGIN
93
94      WITH Start_of_temps^ DO
95      BEGIN
96          READ (Cet, Temperatures[I].Year_number );
97          FOR Period := January TO Annual DO
98              READ(Cet, Temperatures[I].Monthly_temps[Period]);
99          READLN(Cet);
100         I := I + 1;
101     END;
102
103 END;
104
105 CLOSE (Cet);
106
107 WITH Start_of_temps^ DO
108 BEGIN
109     Latest_year := Temperatures[I-1].Year_number;
110     Scale_name  := Celsius;
111     Scale_wanted := Fahrenheit;
112 END;
113
114
115 JOB_PORT ( This_port,
116            STATUS := Returned_status );
117
118     Check_status_and_report ( Returned_status,
119                               'JOB_PORT' );
120
121 CREATE_JOB ( LAB_6As_port,
122             'LAB_6A',
123             '',
124             '',
125             'ENGLISH_TEMPERATURES',
126             NOTIFY := This_port,
127             STATUS := Returned_status );
128
129     Check_status_and_report ( Returned_status,
130                               'CREATE_JOB' );
131
132 WAIT_ANY ( This_port,
133           STATUS := Returned_status );
134
135     Check_status_and_report ( Returned_status,
136                               'WAIT_ANY' );
137
138 END {of PROGRAM}.
139 END {of MODULE};

```

K.18 SOLUTION TO EXERCISE 6 - LAB_6A.PAS

This program connects the circuit with LAB_6

```

1  {-----}
2  SOURCE:      LAB_6A.PAS
3
4  PURPOSE:     This program converts temperatures in an AREA created
5                by LAB_6 to a chosen scale and prints the results
6
7  COMPILE:     $ EPASCAL /LIST /DEBUG LAB_6A, -
8                _$ VAXELN-MODULES /LIBRARY
9
10 LINK:        $ LINK /DEBUG LAB_6A, VAXELN-MODULES /LIBRARY, -
11                _$ ELN$:RTLSHARE /LIBRARY, -
12                _$ ELN$:RTL /LIBRARY /INCLUDE= -
13                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
14                _$ OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
15
16 BUILD:       $ EBUILD /NOEDIT LAB_6
17
18 NOTES:       1) Command procedure LAB_6.COM compiles, links
19                and builds this module into a VAXELN system
20 -----}
21 MODULE Lab_6A [IDENT ('V1.000')];
22
23 INCLUDE
24     Check_status_and_report;
25
26 CONST
27     LF          = '(10);          {line-feed}
28     First_year  = 1659;
29     Last_year   = 2058;
30
31 TYPE
32     Months      = (January,  February, March,   April,
33                    May,       June,    July,     August,
34                    September, October, November, December,
35                    Annual);
36

```

SOLUTIONS TO EXERCISES

```

37      Temperature_scale = (Celsius, Fahrenheit, Kelvin, Reaumur);
38
39      Years_record = RECORD
40          Year_number      : First_year..Last_year;
41          Monthly_temps    : ARRAY [January..Annual] OF INTEGER;
42      END;
43
44      Temperature_table = RECORD
45          Latest_year      : First_year..Last_year;
46          Scale_name       : Temperature_scale;
47          Scale_wanted     : Temperature_scale;
48          Temperatures     : ARRAY [First_year..Last_year] OF Years_record;
49      END;
50
51  VAR
52      Temperature_area: AREA;
53      Year             : First_year..Last_year;
54      Start_of_temps   : ^Temperature_table;
55      Returned_status  : INTEGER;
56
57  { Temperature conversion function definitions here }
58
59      %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'
60
61  { ----- PROGRAM BLOCK ----- }
62
63  PROGRAM Lab_6A (OUTPUT);
64
65  VAR
66      I          : First_year..Last_year;
67      Period     : January..Annual;
68
69  BEGIN
70
71  CREATE_AREA (   Temperature_area,
72                Start_of_temps,
73                'ENGLISH_TEMPERATURES',
74                STATUS := Returned_status );
75
76      Check_status_and_report ( Returned_status,
77                                'CREATE_AREA' );
78  WITH Start_of_temps^ DO
79  BEGIN
80      WRITELN ( ' Temperatures for Central England in degrees ',
81                Scale_wanted, LF );
82      WRITELN ( ' Period: ', First_year:4, ' to ', Latest_year:4,
83                LF, LF);
84

```

```
85   FOR I := First_year TO Latest_year DO
86
87   BEGIN
88     WRITE ( Temperatures[I].Year_number:5 );
89     FOR Period := January TO Annual DO
90       CASE Scale_wanted OF
91
92         Fahrenheit :
93           WRITE ( Celsius_to_Fahrenheit
94                 (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);
95
96         Kelvin      :
97           WRITE ( Celsius_to_Kelvin
98                 (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);
99
100        Reaumur      :
101          WRITE ( Celsius_to_Reaumur
102                (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);
103
104        END; {CASE}
105
106        WRITELN;
107
108      END; {FOR loop}
109    END; {WITH}
110
111  END {of PROGRAM}.
112 END {of MODULE};
```

SOLUTIONS TO EXERCISES

K.18.1 Running LAB_6

```
1 $      ! LAB_6.COM
2 $      ! Command procedure to build the VAXELN modules
3 $      ! LAB_6 and LAB_6A into a system
4 $      !
5 $      ON ERROR THEN GOTO Switch_off_verify
6 $      SET DEFAULT Default_directory
7 $ !
8 $      SET VERIFY
9 $ Compile1:
10 $      EPASCAL -
11          /LIST -
12          /DEBUG LAB_6, VAXELN-MODULES /LIBRARY
13 $ Compile2:
14 $      EPASCAL -
15          /LIST -
16          /DEBUG LAB_6A, VAXELN-MODULES /LIBRARY
17 $ Link1:
18 $      LINK -
19          /DEBUG LAB_6 -
20          ,VAXELN-MODULES /LIBRARY -
21          ,ELN$:RTLSHARE /LIBRARY -
22          ,ELN$:RTL /LIBRARY /INCLUDE= -
23          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
24          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
25 $ Link2:
26 $      LINK -
27          /DEBUG LAB_6A -
28          ,VAXELN-MODULES /LIBRARY -
29          ,ELN$:RTLSHARE /LIBRARY -
30          ,ELN$:RTL /LIBRARY /INCLUDE= -
31          (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
32          OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
33 $ System_build:
34 $      EBUILD -
35          /NOEDIT LAB_6
36 $ Switch_off_verify:
37 $      SET NOVERIFY
38 $      EXIT
```

The .DAT file used by EBUILD looks like this:

```
1      characteristic /noconsole
2      program LAB_6 /debug
3      program LAB_6A /norun /debug
```

Output from LAB_6:

```

SH SYS
! Available: Pages: 17749, Page table slots: 51, Pool blocks: 271
! Time since SET TIME: Idle:  0 00:00:13.39 Total:  0 00:00:13.55
! Time used by past jobs:  0 00:00:00.02
!
! Job 2, program XQDRIVER, priority 1 is waiting.
! Job 3, program EDEBUGREM, priority 3 is running.
! Job 4, program FALSERVER, priority 16 is waiting.
! Job 5, program LAB_6, priority 16 is waiting.
!
GO
! Job 5, process 1, program LAB_6 running.
! Loading traceback data from: DISK$INSTRUCT:[SHONE.VAXELN]LAB_6A.EXE;5
!
! Job 6, process 1, program LAB_6A needs attention.
! Module LAB_6A
! 119:
!>>120: BEGIN
! 121:
! 122: CREATE_AREA (   Temperature_area,
! 123:                Start_of_temps,
! 124:                'ENGLISH_TEMPERATURES',
GO
! Job 6, process 1, program LAB_6A running.
! Temperatures for Central England in degrees FAHRENHEIT
!
! Period: 1659 to 1985
!
!
! 1659 37.4 39.2 42.8 44.6 51.8 55.4 60.8 60.8 55.4 50.0 41.0 35.6 47.8
! 1660 32.0 39.2 42.8 48.2 51.8 57.2 59.0 60.8 55.4 50.0 42.8 41.0 48.4
! 1661 41.0 41.0 42.8 46.4 51.8 57.2 59.0 59.0 55.4 51.8 46.4 42.8 49.5
!
!           <output suppressed to save space>
!
! 1982 36.7 40.5 42.6 47.3 52.7 59.9 61.7 60.3 57.4 49.8 46.0 39.7 49.5
! 1983 43.7 34.9 43.7 43.9 50.0 57.2 66.6 63.0 56.3 50.4 45.1 41.7 49.6
! 1984 37.9 37.8 40.8 46.8 50.0 58.5 62.6 64.2 57.6 52.9 46.8 41.9 49.8
! 1985 34.0 36.1 40.8 47.3 52.0 55.0 61.7 59.4 59.5 52.3 39.7 43.3 48.4
! Job 6, process 1, program LAB_6A has exited.
!
! Job 5, process 1, program LAB_6 running.
! Job 5, process 1, program LAB_6 has exited.
!
exit

```


INDEX

- 11/730
 - setting default bootstrap selection switch, 6-6
 - target console boot command, 6-5
- 11/750
 - setting default bootstrap selection switch, 6-6
 - target console boot command, 6-5
- ::
 - typecast operator, 2-17
- AA-EU40B-TE
 - VAXELN C Run-Time Library Reference Manual, 0-2
- AA-EU41B-TE
 - VAXELN Application Design Guide, 0-2
- AA-GR65B-TE
 - VAX Language-Sensitive Editor VAXELN Pascal Guide, 0-2
- AA-HW72A-TE
 - VAXELN FORTRAN Programmer's Guide, 0-2
- AA-JG87A-TE
 - VAXELN Host System Guide, 0-2
- AA-JL11A-TE
 - VAXELN Introduction To VAXELN, 0-2
- AA-JM81A-TE
 - VAXELN Run-Time Facilities Guide, 0-2
- AA-JN09A-TE
 - VAXELN Pascal Language Reference Manual Part 2, 0-2
- AA-JP29A-TE
 - VAXELN Pascal Language Reference Manual Part 1, 0-2
- AA-Z454F-TE
 - VAXELN Release Notes, 0-2
- Abbreviations
 - student guide, 0-2
- Abort
 - exception type defined, 15-2
 - type of hardware exception, 15-1
- Absolute time
 - format, 9-6
- ACCEPT_CIRCUIT
 - predeclared procedure, 3-8, 10-11 to 10-12
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_9.PAS, E-56
 - used in LAB_5.PAS, K-68
 - Access control string, 14-2
- ACCESS\$DIRECT
 - value of predeclared enumerated type FILE_ACCESS, 2-14
- ACCESS\$SEQUENTIAL
 - value of predeclared enumerated type FILE_ACCESS, 2-14
- Ada
 - supported on VAXELN, 0-1
- ADAWI
 - VAX instruction
 - used by ADD_INTERLOCKED, 2-19
- ADD_INTERLOCKED
 - predeclared function
 - use in sharing data, 2-19
- ADDRESS
 - illegal use of, 2-18
 - predeclared function, 2-18
- Addresses of variables, 2-18
- ADG
 - abbreviation for VAXELN Application Design Guide, 0-2
 - Application Design Guide, 6-2
- AGGREG_1.PAS, B-39
- Aggregate initialization
 - example
 - AGGREG_1.PAS, B-39
- Aggregate initializers
 - use on arrays and records, 2-13
- ALIGNED
 - attribute
 - alignment values, 2-10
 - extent expression, 2-10
 - use in declarations of variables, 2-15
 - use on field of record, 2-8
 - use with arrays, 2-8
 - used on field of record, 2-10
 - used on record, 2-8, 2-10
 - data size attribute, 2-10
- ANYTYPE
 - data type

- allowed as pointers
 - associated data type, 2-8
 - invalid for message pointer, 10-7
- AP
 - abbreviation for Argument Pointer, 15-5
- Application
 - simple, 1-2
- Application Design Guide
 - VAXELN, 0-1
- AREA
 - creating new, 10-5
 - definition, 10-5
 - mapping existing, 10-5
 - object for WAIT procedure, 9-3
 - satisfying wait for, 9-3
 - system data type, 2-9
- Area code
 - part of node address, 6-3
- Area communications
 - example
 - COMM_5.PAS, E-28
- AREA object, 3-1
 - binary semaphore, 3-2
 - details, 3-2
 - name facility, 3-2
 - operations on, 3-2
 - pool space consumption, 3-2
 - region allocation, 3-2
 - region mapping, 3-2
 - shareable memory region, 3-2
 - state, 3-2
 - use of P0 space, 3-2
- AREA type
 - parameter to subprocess, 10-2
- ARGUMENT
 - predeclared function, 2-16
 - use in pseudo variable references, 2-16
- Argument(s)
 - EBUILD Program description entry, 5-3
- Arguments
 - CREATE_JOB limitations, 10-4
- Arithmetic operators, 2-20
- Array
 - initialization, 2-13
 - use of extent expressions in bounds declarations, 2-7
- Array type, 2-8
 - as definition of flexible type, 2-7
 - limit on dimension, 2-8
- ASCII codes, A-1
- ASSERT_CHECK_ENABLED
 - predeclared named constant, 2-13
- ATTRIBS_1.PAS, B-20
- Attributes
 - data size, 2-10
 - use in module headers, 2-3
- Authorization
 - destination, 14-2
 - including in system, 14-4
 - proxy, 14-2
 - size of fields in record, 14-6
- Authorization facilities, 14-2
- Authorization file
 - EBUILD Network Node
 - characteristics entry, 5-5
 - entry in EBUILD, 14-6
- Authorization required
 - EBUILD Network Node
 - characteristics entry, 5-5
 - entry in EBUILD, 14-5
- Authorization service
 - EBUILD Network Node
 - characteristics entry, 5-5
 - entry in EBUILD, 14-5
- AUTHORIZE.DAT
 - default authorization file, 14-6
- Autoload driver
 - EBUILD Device description entry, 5-3
- %B
 - radix specifier
 - binary, 2-11
- B DDA0
 - target console boot command 11/750, 6-5
- B XE0
 - target console boot command 11/730, 6-5
- B XQA0
 - target console boot command MicroVAX, 6-5
- BIN function
 - example
 - ATTRIBS_1.PAS, B-20
 - RADIX_1.PAS, B-18
 - TYPECAST_1.PAS, B-23
- Binary radix specifier
 - %B, 2-11
- Binary semaphore
 - AREA object, 3-2
- BIT
 - attribute

- invalid for VALUE attributed variables, 2-16
- use in declarations of variables, 2-15
- data size attribute, 2-5, 2-10
- extent expression, 2-10
- 64-bit
 - internal binary time form, 9-6
- 32-bit limit
 - on VALUE attributed variables, 2-16
- Boot method
 - EBUILD System characteristics entry, 5-4
- Bootable
 - Files-11 disk, making, 6-1
 - Floppy disk, making, 6-2
 - making media, 6-1
 - TU58, making, 6-1
- Booting and downline loading, 6-1
- Booting from disk, 6-1
- Bootstrap image file
 - NEWBOOT procedure for loading, 6-4
- Bootstrap loaders
 - configuring, 6-4
- BOTH
 - Debug option, 7-2
- Bound flexible type, 2-7
 - pointer to as definition of flexible type, 2-6
 - use in typecasting, 2-17
- BRIEF
 - EBUILD qualifier, 5-2
- Building a VAXELN system, 1-7
- BYTE
 - attribute
 - use in declarations of variables, 2-15
 - data size attribute, 2-6, 2-10
- BYTE_DATA
 - data type, 2-9
 - example
 - ATTRIBS_1.PAS, B-20
 - flexible data type, 2-6
- C - language
 - supported on VAXELN, 0-1
- C Run-Time Library Reference Manual
 - VAXELN, 0-1
- CALL
 - EDEBUG command, 7-4
- Call frame, 15-6
- CANCEL BREAK
 - EDEBUG command, 7-4
- CANCEL CONTROL
 - EDEBUG command, 7-4
- CANCEL EXCEPTION BREAK
 - EDEBUG command, 7-4
- CARRIAGE\$FORTRAN
 - value of predeclared enumerated type FILE_CARRIAGE_CONTROL, 2-14
- CARRIAGE\$LIST
 - value of predeclared enumerated type FILE_CARRIAGE_CONTROL, 2-14
- CARRIAGE\$NONE
 - value of predeclared enumerated type FILE_CARRIAGE_CONTROL, 2-14
- CASE
 - constant range limitation, 2-22
- CASE statement, 2-21
 - OTHERWISE clause in, 2-22
- CHECK
 - EPASCAL qualifier, 4-2
- CHECK=ASSERT
 - EPASCAL command qualifier, 2-13
- CHECK_STATUS_AND_REPORT.PAS
 - error reporting routine, C-1
- CHF\$R_MECH_ARGS
 - predeclared type, 15-9
- CHF\$R_SIGNAL_ARGS
 - predeclared type, 15-9
- CHF\$R_SIGNAL_ARGS_ADDITIONAL
 - predeclared type, 15-9
- Circuit and expedited message
 - example
 - COMM_9.PAS, E-56
- Circuit communication
 - example
 - COMM_7.PAS, E-39
 - COMM_7A.PAS, E-42
 - COMM_8.PAS, E-46
 - COMM_8A.PAS, E-50
 - COMM_9A.PAS, E-60
- CIRCUIT\$ACCEPT
 - value of predeclared enumerated type OPEN_CIRCUIT, 2-14
- CIRCUIT\$CONNECT
 - value of predeclared enumerated type OPEN_CIRCUIT, 2-14
- Circuits
 - advantages of, 10-11
 - disconnecting, 10-14
 - message transmission method, 10-10
 - using, 10-11

- CLEAR
 - state of EVENT object, 3-5
- CLEAR NODE
 - NCP command, 6-7
- CLEAR_EVENT
 - predeclared procedure, 3-5, 9-4
- CLOSE
 - predeclared file routine, 12-1
 - use in communications, 11-11
- CMKRNL
 - privilege required to use
 - NEWBOOT.COM, 6-4
- COMM_1.PAS, E-1
- COMM_2.PAS, E-7
- COMM_3.PAS, E-14
- COMM_3A.PAS, E-17
- COMM_4.PAS, E-23
- COMM_5.PAS, E-28
- COMM_6.PAS, E-34
- COMM_6A.PAS, E-36
- COMM_7.PAS, E-39
- COMM_7A.PAS, E-42
- COMM_8.PAS, E-46
- COMM_8A.PAS, E-50
- COMM_9.PAS, E-56
- COMM_9A.PAS, E-60
- Communication
 - as provided by kernel, 1-2
- Communication between jobs, 10-1
- Communication region, 13-3
- Communication techniques
 - examples, E-1
- Compilation
 - command EPASCAL, 4-1, 5-1
- Compilation unit, 8-3
- Compiling
 - simple program, 1-7
- Compiling VAXELN sources, 4-1
- Compound statement
 - part of routine body, 8-2
- Concatenation
 - operator for strings, 2-21
- Concatenation operator, 2-21
- Condition
 - definition of, 15-3
- Condition values, 15-3
 - layout of 32-bit pattern, 15-3
- Configuring bootstrap loaders, 6-4
- Connect time
 - EBUILD System characteristics
 - entry, 5-4
- CONNECT_CIRCUIT
 - connecting with VMS, 11-10
 - predeclared procedure, 3-8, 10-11 to 10-12
 - used in COMM_7A.PAS, E-42
 - used in COMM_8A.PAS, E-50
 - used in COMM_9A.PAS, E-60
 - used in LAB_5A.PAS, K-72
 - used in TIME_3.PAS, D-19
- CONNECTED
 - state of PORT object, 3-8
- Connecting to and from VMS, 11-10
- Connecting to VAXELN from VMS, 11-11
- Connecting to VMS, 11-10
- Console
 - EBUILD System characteristics
 - entry, 5-4
- Console mode, 1-9
- CONST
 - declaration in MODULE, 8-3
 - use of extent expressions in, 2-7
- Constants, 2-11
 - named
 - typecasting not allowed on, 2-18
 - named, predeclared, 2-13
 - ordinal
 - exporting, 2-4
 - storage allocation, 2-18
- Continuing
 - action of exception handler, 15-12
- Control and status register, 13-3
- Control-C session
 - EDEBUG, 7-4
 - command restrictions, 7-4
 - prompt
 - EDEBUG, 7-4
- Conventions used
 - student guide, 0-2
- COPY_FILE
 - predeclared file utility, 12-2
- Course
 - documentation, 0-1
 - objectives, 0-1
 - Prerequisites, 0-1
 - Previous training, 0-1
- CREATE JOB
 - EDEBUG command, 7-4
- CREATE PROCESS
 - EDEBUG command, 7-4
- CREATE_AREA
 - details of service, 10-5
 - predeclared procedure, 3-2, 10-5

- call format, 3-3
- used in COMM_5.PAS, E-28
- used in LAB_4.PAS, K-56
- used in LAB_4A.PAS, K-58
- used in LAB_4B.PAS, K-62
- used in LAB_6.PAS, K-78
- used in LAB_6A.PAS, K-81
- CREATE_DEVICE
 - kernel service for attaching
 - ISR to device, 2-2
 - predeclared procedure, 3-3
 - call format, 3-4, 13-4
- CREATE_DIRECTORY
 - predeclared file utility, 12-2
 - predeclared procedure
 - used in EXCEPT_1.PAS, I-1
 - used in FILE_1.PAS, G-1
- CREATE_EVENT
 - predeclared procedure, 3-5, 9-4
 - call format, 3-5
 - used in COMM_2.PAS, E-7
 - used in ERRORS_1.PAS, C-10
 - used in ERRORS_2.PAS, C-13
 - used in NET_1.PAS, F-1
 - used in SYNCH_2.PAS, D-32
 - used in SYNCH_3.PAS, D-36
 - used in SYNCH_4.PAS, D-41
- CREATE_JOB
 - for specifying program
 - arguments, 2-5
 - NOTIFY parameter, 2-5
 - predeclared procedure
 - used in COMM_6.PAS, E-34
 - used in LAB_3.PAS, K-47
 - used in LAB_4.PAS, K-56
 - used in LAB_4A.PAS, K-58
 - used in LAB_6.PAS, K-78
 - used in NET_1.PAS, F-1
 - used in TIME_2.PAS, D-7
 - used in TIME_3.PAS, D-19
 - procedure for creating jobs
 - dynamically, 8-1
 - program arguments with, 10-4
- CREATE_MESSAGE
 - details of service, 10-7
 - predeclared procedure, 3-6,
 - 10-7
 - call format, 3-6
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-14,
 - E-17
 - used in COMM_4.PAS, E-23
 - used in COMM_7A.PAS, E-42
 - used in COMM_8.PAS, E-46
 - used in COMM_8A.PAS, E-50
 - used in COMM_9.PAS, E-56
 - used in COMM_9A.PAS, E-60
 - used in LAB_3.PAS, K-47
 - used in LAB_4.PAS, K-56
 - used in LAB_4A.PAS, K-58
 - used in LAB_5.PAS, K-68
 - used in LAB_5A.PAS, K-72
- CREATE_MUTEX
 - predeclared procedure, 9-5
 - used in SYNCH_5.PAS, D-45
 - used in SYNCH_6.PAS, D-52
- CREATE_NAME
 - details of service, 10-8
 - for naming message ports, 10-8
 - network service response, 11-5
 - predeclared procedure, 3-7
 - call format, 3-7
 - used in COMM_3.PAS, E-17
 - used in COMM_4.PAS, E-23
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_9.PAS, E-56
 - used in LAB_5.PAS, K-68
- CREATE_PORT
 - details of service, 10-8
 - predeclared procedure, 3-8,
 - 10-7
 - call format, 3-9
 - used in COMM_7.PAS, E-39
 - used in COMM_7A.PAS, E-42
 - used in COMM_8.PAS, E-46
 - used in COMM_8A.PAS, E-50
 - used in COMM_9.PAS, E-56
 - used in COMM_9A.PAS, E-60
 - used in LAB_3.PAS, K-47
 - used in LAB_4.PAS, K-56
 - used in LAB_4A.PAS, K-58
 - used in LAB_5.PAS, K-68
 - used in LAB_5A.PAS, K-72
- CREATE_PROCESS
 - kernel service for invoking
 - process block, 2-2, 8-1
 - predeclared procedure, 3-9
 - call format, 3-10
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_5.PAS, E-28
 - used in LAB_2.PAS, K-20
 - used in LAB_2A.PAS, K-27
 - used in LAB_2B.PAS, K-34
 - used in MOD-PROG-PROC-1.PAS,
 - C-3
 - used in SYNCH_1.PAS, C-6
 - used in SYNCH_2.PAS, D-32
 - used in SYNCH_3.PAS, D-36
 - used in SYNCH_4.PAS, D-41
 - used in SYNCH_5.PAS, D-45

- used in SYNCH_6.PAS, D-52
- CREATE_SEMAPHORE
 - predeclared procedure, 3-11, 9-4
 - call format, 3-11
- CROSS_REFERENCE
 - EPASCAL qualifier, 4-2
- CSR
 - abbreviation for control and status register, 13-3
- CTRL-C
 - EDEBUG command, 7-4
- CTRL-Z
 - EDEBUG command, 7-5
- CTRLERR XQAO
 - console message, 6-8
- CURRENT_PROCESS
 - predeclared procedure, 3-9
- .DAT file
 - produced by EBUILD, 1-8
 - use with EBUILD, 5-2
- Data
 - inaccessible process, 2-18
 - PSECT, 2-18
 - shared
 - initialization of, 2-20
 - sharing between processes, 2-18
- Data size attribute
 - BIT, 2-5
 - BYTE, 2-6
 - LONG, 2-6
 - WORD, 2-6
- Data types, 2-5
 - miscellaneous predeclared, 2-9
- datagram communications
 - example
 - COMM_1.PAS, E-1
 - COMM_2.PAS, E-7
 - COMM_3.PAS, E-14
 - COMM_3A.PAS, E-17
 - COMM_4.PAS, E-23
- Datagrams
 - disadvantages of, 10-11
 - message transmission method, 10-10
- DBG\$INPUT
 - logical name used with EDEBUG, 7-3
- DBG\$OUTPUT
 - logical name used with EDEBUG, 7-3
- DDCMP
 - DIGITAL Data Communications Message Protocol, 5-6
- EBUILD Terminal descriptions
 - entry, 5-6
- DEBUG
 - EDEBUG command, 7-5
 - EPASCAL command qualifier, 1-7
 - EPASCAL qualifier, 4-2
 - LINK command qualifier, 1-7
- Debug
 - EBUILD Program description
 - entry, 5-3
 - EBUILD System characteristics
 - entry, 5-4
- Debugger
 - local
 - memory required, 1-4
 - remote
 - memory required, 1-4
- Debugging, 7-1
 - choosing a mode, 7-2
 - locally, 7-1
 - methods, 7-1
 - remotely, 7-1
 - use of /DEBUG qualifier when compiling, 7-2
 - use of /DEBUG qualifier when linking, 7-2
- Declarations
 - part of routine body, 8-2
- DECnet, 11-1
 - message size, 11-2
 - Phase IV, 11-1
 - target system licence, 1-5
- DECnet-VAX
 - requirement for downline loading, debugging and communications, 1-5
- Default source file type, 4-1
- Default UIC
 - EBUILD Network Node
 - characteristics entry, 5-5
- DEFINE
 - EDEBUG command, 7-5
 - NCP command requiring privilege, 6-4
- DELETE
 - predeclared procedure, 3-2, 3-4 to 3-7, 3-9 to 3-11, 9-4
 - used in ERRORS_1.PAS, C-10
 - used in ERRORS_2.PAS, C-13
 - used in LAB_3.PAS, K-47
- DELETE PROCESS
 - EDEBUG command, 7-5
- DELETE_FILE
 - predeclared file utility, 12-2
 - predeclared procedure

- used in EXCEPT_1.PAS, I-1
 - used in FILE_1.PAS, G-1
- DELETE_MUTEX
 - predeclared procedure, 9-5
- DELETE_NAME
 - network service response, 11-5
- Delta time
 - format, 9-7
- DEPOSIT
 - EDEBUG command, 7-5
 - restriction on use, 7-4
- DEQNA
 - abbreviation for DIGITAL Ethernet to Q-bus network adapter, 1-4
 - Ethernet adapter at target MicroVAX, 1-4
 - possible problem booting from, 6-8
- Destination authorization, 14-2
- DEUNA
 - abbreviation for DIGITAL Ethernet to Unibus network adapter, 1-4
 - Ethernet adapter at target VAX, 1-4
- Developing a simple application, 1-5
- Development system
 - hardware required, 1-3
 - Memory and space required, 1-4
 - software required, 1-5
- DEVICE
 - object for WAIT procedure, 9-3
 - satisfying wait for, 9-3
 - system data type, 2-9
- Device
 - definition, 13-1
 - initialization routines, 13-7
- Device drivers
 - definition, 13-1
 - DIGITAL supplied programs, 1-2
 - examples, H-1
 - memory required, 1-4
 - supplied with VAXELN, 13-2
- DEVICE ISR
 - dispatcher
 - pool space required, 3-3
- DEVICE object, 3-1
 - details, 3-3
 - operations on, 3-3
 - pool space consumption, 3-3
- DEVICE type
 - parameter to subprocess, 10-2
- Devices, drivers and interrupts, 13-1
- DIFFERENCES
 - DCL command used remotely from VMS, 12-8
- Directives
 - procedures and functions, 2-22
- DIRECTORY_CLOSE
 - predeclared file utility, 12-2
- DIRECTORY_LIST
 - predeclared file utility, 12-2
 - predeclared procedure
 - used in EXCEPT_1.PAS, I-1
 - used in FILE_1.PAS, G-1
- DIRECTORY_OPEN
 - predeclared file utility, 12-2
 - predeclared procedure
 - used in EXCEPT_1.PAS, I-1
 - used in FILE_1.PAS, G-1
- DISABLE_INTERRUPT
 - predeclared routine, 13-4
 - requires kernel mode, 13-4
- DISCONNECT_CIRCUIT
 - predeclared procedure, 3-8
 - used in TIME_3.PAS, D-19
 - procedure, 10-14
- Disk space
 - required for development system, 1-4
- DISK\$name
 - label form for disk volumes, 12-5
- Disk/volume names
 - EBUILD System characteristics
 - entry, 5-4
- DISMOUNT_TAPE_VOLUME
 - predeclared tape utility, 12-2
- DISMOUNT_VOLUME
 - predeclared disk utility, 12-2
- Display type
 - EBUILD Console characteristics
 - entry, 5-6
 - EBUILD Terminal descriptions
 - entry, 5-6
- DISPOSITION\$DELETE
 - value of predeclared enumerated type FILE_DISPOSITION, 2-14
- DISPOSITION\$SAVE
 - value of predeclared enumerated type FILE_DISPOSITION, 2-14
- Documentation, 0-1
- DOUBLE
 - data type
 - use in exponentiation, 2-20
- Down-line loading

- simple system, 1-9
- Downline loader
 - 11/730
 - TU58 console medium, 6-4
 - 11/750
 - TU58 console medium, 6-4
 - MicroVAX
 - Boot ROM, 6-4
- Downline loading, 6-5
 - problems, 6-7
- Dyadic operators
 - use with LARGE_INTEGER, 2-20
- Dynamic program space
 - EBUILD System characteristics
 - entry, 5-4
- EBUILD
 - command invoking the System
 - Builder, 5-1
 - .DAT file, 5-2
 - default qualifiers, 5-1
 - files produced, 1-8
 - menus
 - Console characteristics, 5-6
 - Device descriptions, 5-3
 - Network Node characteristics, 5-5
 - Program descriptions, 5-3
 - System characteristics, 5-4
 - Terminal descriptions, 5-6
 - stage in development, 1-5
 - system builder command line, 1-7
 - VAXELN system builder command, 1-5
- EBUILD /NOEDIT, 5-2
- EBUILD Console characteristics
 - entry
 - Display type, 5-6
 - Echo, 5-6
 - Eight-bit, 5-6
 - Escape recognition, 5-6
 - Pass all, 5-6
- EBUILD Device description entry
 - Autoload driver, 5-3
 - Interrupt priority, 5-3
 - Name, 5-3
 - Register address, 5-3
 - Vector address, 5-3
- EBUILD map
 - example of brief, 5-7
 - example of full, 5-10
- EBUILD Network Node
 - characteristics entry
 - Authorization file, 5-5

- Authorization required, 5-5
- Authorization service, 5-5
- Default UIC, 5-5
- File Access Listener, 5-5
- Name server, 5-5
- Network device, 5-5
- Network segment size, 5-5
- Network Service, 5-5
- Node address, 5-5
- Node name, 5-5
- Node triggerable, 5-5
- EBUILD Program description entry
 - Argument(s), 5-3
 - Debug, 5-3
 - Init required, 5-3
 - Job port message limit, 5-3
 - Job priority, 5-3
 - Kernel stack, 5-3
 - Mode, 5-3
 - Powerfailure exception, 5-3
 - Process priority, 5-3
 - Run, 5-3
 - User stack, 5-3
- EBUILD System characteristics
 - entry
 - Boot method, 5-4
 - Connect time, 5-4
 - Console, 5-4
 - Debug, 5-4
 - Disk/volume names, 5-4
 - Dynamic program space, 5-4
 - Guaranteed image list, 5-4
 - I/O region size, 5-4
 - Instruction emulation, 5-4
 - Interrupt stack, 5-4
 - Memory limit, 5-4
 - Page table slots, 5-4
 - Pool size, 5-4
 - Ports, 5-4
 - System image, 5-4
 - Time interval, 5-4
 - Virtual size, 5-4
- EBUILD Terminal descriptions
 - entry
 - DDCMP, 5-6
 - Display type, 5-6
 - Echo, 5-6
 - Eight-bit, 5-6
 - Escape recognition, 5-6
 - Modem, 5-6
 - Parity, 5-6
 - Parity type, 5-6
 - Pass all, 5-6
 - Speed, 5-6
 - Terminal, 5-6

- Terminal type, 5-6
- Echo
 - EBUILD Console characteristics entry, 5-6
 - EBUILD Terminal descriptions entry, 5-6
- EDEBUG
 - command and qualifiers, 7-2
 - command restrictions
 - Control-C session, 7-4
 - commands, 7-4
 - Control-C session, 7-4
 - Control-C session prompt, 7-4
 - error messages, 7-3
 - facilities, 7-3
 - logical names used with, 7-3
 - prompt, 7-3
 - remote debugging tool, 7-1
 - requires OPER privilege, 6-5
 - session states, 7-4
 - use for loading system, 6-5
 - use in downline loading, 1-9
 - use of NODEBUG qualifier, 7-3
- EDEBUG command
 - CALL, 7-4
 - CANCEL BREAK, 7-4
 - CANCEL CONTROL, 7-4
 - CANCEL EXCEPTION BREAK, 7-4
 - CREATE JOB, 7-4
 - CREATE PROCESS, 7-4
 - CTRL-C, 7-4
 - CTRL-Z, 7-5
 - DEBUG, 7-5
 - DEFINE, 7-5
 - DELETE PROCESS, 7-5
 - DEPOSIT, 7-5
 - EVALUATE, 7-5
 - EXAMINE, 7-5
 - EXIT, 7-5
 - GO, 7-5
 - HALT, 7-5
 - HELP, 7-5
 - IF, 7-5
 - LEAVE, 7-5
 - LOAD, 7-5
 - PREDECESSOR, 7-5
 - SEARCH, 7-5
 - SET BREAK, 7-5
 - SET COMMAND, 7-6
 - SET CONTROL, 7-6
 - SET EXCEPTION BREAK, 7-6
 - SET LOG, 7-6
 - SET MODE, 7-6
 - SET PROGRAM, 7-6
 - SET RETURN BREAK, 7-6

- SET SESSION, 7-6
- SET STEP, 7-6
- SET TIME, 7-6
- SHOW BREAK, 7-6
- SHOW CALLS, 7-6
- SHOW COMMAND, 7-6
- SHOW JOB, 7-6
- SHOW MESSAGE, 7-6
- SHOW MODE, 7-6
- SHOW MODULE, 7-6
- SHOW PROCESS, 7-6
- SHOW PROGRAM, 7-6
- SHOW SESSION, 7-7
- SHOW STEP, 7-7
- SHOW SYMBOL, 7-7
- SHOW SYSTEM, 7-7
- SHOW TIME, 7-7
- SHOW TRANSLATION, 7-7
- SUCCESSOR, 7-7
- TYPE, 7-7
- UNLOAD, 7-7
- EDEBUG-F-COMMERROR
 - error message from EDEBUG, 7-3
- EDEBUG-F-NOCONNECT
 - error message from EDEBUG, 7-3
- EDIT
 - DCL command used remotely from VMS, 12-8
 - EBUILD qualifier, 5-2
- EDT
 - text editor, 1-6
- Eight-bit
 - EBUILD Console characteristics entry, 5-6
 - EBUILD Terminal descriptions entry, 5-6
- ELN\$:CRTLSHARE/LIB
 - input to linker, 1-7
- ELN\$:FRTLOBJECT/LIB
 - input to linker, 1-7
- ELN\$:LPVDRIVER.PAS
 - source listing, H-1
- ELN\$:RTL.OLB, 4-4
- ELN\$:RTL/LIB
 - input to linker, 1-7
- ELN\$:RTLSHARE.OLB, 4-4
- ELN\$:RTLSHARE/LIB
 - input to linker, 1-7
- ELN\$:VAXELNC/LIB
 - input to C compiler, 1-7
- ELN\$AUTH_ADD_USER
 - predeclared authorization procedure, 14-6
- ELN\$AUTH_MODIFY_USER

- predeclared authorization procedure, 14-6
- ELN\$AUTH_REMOVE_USER
 - predeclared authorization procedure, 14-6
- ELN\$AUTH_SHOW_USER
 - predeclared authorization procedure, 14-6
- ELN\$GET_STATUS_TEXT
 - predeclared procedure, 15-14
 - used in ERRORS_2.PAS, C-13
 - used in EXCEPT_1.PAS, I-1
 - used in FILE_1.PAS, G-1
- ELN\$LOAD_PROGRAM
 - predeclared procedure
 - used in NET_1.PAS, F-1
- ELN_COMPILE_1.COM
 - listing, B-4
- ELN_EBUILD_1.COM
 - listing, B-4
- ELN_LINK_1.COM
 - listing, B-4
- End-node routing, 11-1
- Enumerated types, 2-6
 - additional for file handling, 12-4
 - limit on values, 2-6
 - predeclared, 2-14
- EOF
 - predeclared file routine, 12-1
- EOLN
 - Predeclared text-file routine, 12-1
- EPASCAL
 - command qualifier
 - CHECK=ASSERT, 2-13
 - compilation command, 4-1
 - default compilation qualifiers, 4-1
 - qualifiers overriding %INCLUDE options, 2-1
- Error
 - severity code value 2, 15-4
- Error reporting
 - example
 - ERRORS_1.PAS, C-10
 - ERRORS_2.PAS, C-13
- ERRORS_1.PAS, C-10
- ERRORS_2.PAS, C-13
- Escape recognition
 - EBUILD Console characteristics entry, 5-6
 - EBUILD Terminal descriptions entry, 5-6
- ESTABLISH
 - predeclared procedure, 15-14
- Ethernet address
 - use in name server selection, 11-6
- Ethernet hardware address, 6-3
- EVALUATE
 - EDEBUG command, 7-5
- EVE
 - abbreviation for Extensible VAX Editor, 1-6
 - text editor, 1-6
- EVENT
 - object for WAIT procedure, 9-3
 - satisfying wait for, 9-3
 - system data type, 2-9
- EVENT object, 3-1
 - details, 3-5
 - operations on, 3-5
 - pool space consumption, 3-5
 - state, 3-5
- EVENT type
 - parameter to subprocess, 10-2
- EVENT\$CLEARED
 - value of predeclared enumerated type EVENT_STATE, 2-14
- EVENT\$SIGNALLED
 - value of predeclared enumerated type EVENT_STATE, 2-14
- EVENT_STATE
 - predeclared enumerated type, 2-14
- EXAMINE
 - EDEBUG command, 7-5
 - restriction on use, 7-4
- Example
 - ACCEPT_CIRCUIT
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_9.PAS, E-56
 - used in LAB_5.PAS, K-68
 - aggregate initialization
 - AGGREG_1.PAS, B-39
 - area communications
 - COMM_5.PAS, E-28
 - BIN function
 - ATTRIBS_1.PAS, B-20
 - RADIX_1.PAS, B-18
 - TYPECAST_1.PAS, B-23
 - BYTE_DATA
 - ATTRIBS_1.PAS, B-20
 - Circuit and expedited message
 - COMM_9.PAS, E-56
 - Circuit communication
 - COMM_7.PAS, E-39
 - COMM_7A.PAS, E-42

COMM_8.PAS, E-46
 COMM_8A.PAS, E-50
 COMM_9A.PAS, E-60
 CONNECT_CIRCUIT
 used in COMM_7A.PAS, E-42
 used in COMM_8A.PAS, E-50
 used in COMM_9A.PAS, E-60
 used in LAB_5A.PAS, K-72
 used in TIME_3.PAS, D-19
 CREATE_AREA
 used in COMM_5.PAS, E-28
 used in LAB_4.PAS, K-56
 used in LAB_4A.PAS, K-58
 used in LAB_4B.PAS, K-62
 used in LAB_6.PAS, K-78
 used in LAB_6A.PAS, K-81
 CREATE_DIRECTORY
 used in EXCEPT_1.PAS, I-1
 used in FILE_1.PAS, G-1
 CREATE_EVENT
 used in COMM_2.PAS, E-7
 used in ERRORS_1.PAS, C-10
 used in ERRORS_2.PAS, C-13
 used in NET_1.PAS, F-1
 used in SYNCH_2.PAS, D-32
 used in SYNCH_3.PAS, D-36
 used in SYNCH_4.PAS, D-41
 CREATE_JOB
 used in COMM_6.PAS, E-34
 used in LAB_3.PAS, K-47
 used in LAB_4.PAS, K-56
 used in LAB_4A.PAS, K-58
 used in LAB_6.PAS, K-78
 used in NET_1.PAS, F-1
 used in TIME_2.PAS, D-7
 used in TIME_3.PAS, D-19
 CREATE_MESSAGE
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-14,
 E-17
 used in COMM_4.PAS, E-23
 used in COMM_7A.PAS, E-42
 used in COMM_8.PAS, E-46
 used in COMM_8A.PAS, E-50
 used in COMM_9.PAS, E-56
 used in COMM_9A.PAS, E-60
 used in LAB_5.PAS, K-68
 used in LAB_5A.PAS, K-72
 CREATE_MUTEX
 used in SYNCH_5.PAS, D-45
 used in SYNCH_6.PAS, D-52
 CREATE_NAME
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23

 used in COMM_7.PAS, E-39
 used in COMM_8.PAS, E-46
 used in COMM_9.PAS, E-56
 used in LAB_5.PAS, K-68
 CREATE_PORT
 used in COMM_7.PAS, E-39
 used in COMM_7A.PAS, E-42
 used in COMM_8.PAS, E-46
 used in COMM_8A.PAS, E-50
 used in COMM_9.PAS, E-56
 used in COMM_9A.PAS, E-60
 used in LAB_3.PAS, K-47
 used in LAB_4.PAS, K-56
 used in LAB_4A.PAS, K-58
 used in LAB_5.PAS, K-68
 used in LAB_5A.PAS, K-72
 CREATE_PROCESS
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_5.PAS, E-28
 used in LAB_2.PAS, K-20
 used in LAB_2A.PAS, K-27
 used in LAB_2B.PAS, K-34
 used in MOD-PROG-PROC-1.PAS,
 C-3
 used in SYNCH_1.PAS, C-6
 used in SYNCH_2.PAS, D-32
 used in SYNCH_3.PAS, D-36
 used in SYNCH_4.PAS, D-41
 used in SYNCH_5.PAS, D-45
 used in SYNCH_6.PAS, D-52
 datagram communications
 COMM_1.PAS, E-1
 COMM_2.PAS, E-7
 COMM_3.PAS, E-14
 COMM_3A.PAS, E-17
 COMM_4.PAS, E-23
 DELETE
 used in ERRORS_1.PAS, C-10
 used in ERRORS_2.PAS, C-13
 used in LAB_3.PAS, K-47
 DELETE_FILE
 used in EXCEPT_1.PAS, I-1
 used in FILE_1.PAS, G-1
 DIRECTORY_LIST
 used in EXCEPT_1.PAS, I-1
 used in FILE_1.PAS, G-1
 DIRECTORY_OPEN
 used in EXCEPT_1.PAS, I-1
 used in FILE_1.PAS, G-1
 DISCONNECT_CIRCUIT
 used in TIME_3.PAS, D-19
 ELN\$GET_STATUS_TEXT
 used in ERRORS_2.PAS, C-13
 used in EXCEPT_1.PAS, I-1

used in FILE_1.PAS, G-1
 ELN\$LOAD_PROGRAM
 used in NET_1.PAS, F-1
 Error reporting
 ERRORS_1.PAS, C-10
 ERRORS_2.PAS, C-13
 EXPORT
 EXPORT_1.PAS, B-10
 extent expression
 ATTRIBS_1.PAS, B-20
 FUNCTION_TYPE
 used in LAB_1.PAS, K-1
 used in LAB_2.PAS, K-20
 used in LAB_2A.PAS, K-27
 used in LAB_2B.PAS, K-34
 GET_TIME
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-14
 used in SYNCH_1.PAS, D-29
 used in SYNCH_6.PAS, D-52
 used in TIME_1.PAS, D-1
 HEX function
 RADIX_1.PAS, B-18
 IDENT
 SIMPLE.PAS, B-1
 IMPORT
 IMPORT_1.PAS, B-8
 %INCLUDE
 INCLUDE_SOURCE.PAS, B-5
 INCLUDE
 INCLUDE_1.PAS,
 INCLUDE_1_DEFS.PAS, B-13
 INIT_VOLUME
 used in FILE_1.PAS, G-1
 INITIALIZATION_DONE
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23
 used in COMM_7.PAS, E-39
 used in COMM_8.PAS, E-46
 used in COMM_9.PAS, E-56
 used in LAB_5.PAS, K-68
 used in TIME_2A.PAS, D-15
 internal time values
 TIME_4.PAS, D-25
 Job argument data communication
 COMM_6.PAS, E-34
 COMM_6A.PAS, E-36
 JOB_PORT
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-14,
 E-17
 used in COMM_4.PAS, E-23
 used in LAB_6.PAS, K-78

used in TIME_3.PAS, D-19
 KER\$CREATE_PROCESS
 used in LAB_2_C.C, K-40
 KER\$WAIT_ANY
 used in LAB_2_C.C, K-40
 LOCK_MUTEX
 used in SYNCH_5.PAS, D-45
 used in SYNCH_6.PAS, D-52
 module, program and process
 MOD-PROG-PROC-1.PAS, C-3
 MOD-PROG-PROC-1A.PAS, C-6
 MOUNT_VOLUME
 used in FILE_1.PAS, G-1
 MUTEX
 SYNCH_5.PAS, D-45
 SYNCH_6.PAS, D-52
 Non-printing characters in
 constants
 EXPORT_1.PAS, B-10
 OCT function
 RADIX_1.PAS, B-18
 PROGRAM_ARGUMENT
 used in COMM_4.PAS, E-23
 used in COMM_6A.PAS, E-36
 Radix specifiers
 RADIX_1.PAS, B-18
 RECEIVE
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23
 used in COMM_7.PAS, E-39
 used in COMM_8.PAS, E-46
 used in COMM_8A.PAS, E-50
 used in COMM_9A.PAS, E-60
 used in LAB_5.PAS, K-68
 used in LAB_5A.PAS, K-72
 used in TIME_3.PAS, D-19
 Scope
 SCOPE_1.PAS, B-27
 SCOPE_2.PAS, B-29
 SCOPE_3.PAS, B-31
 SCOPE_3A.PAS, B-33
 SEND
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-14
 used in COMM_7A.PAS, E-42
 used in COMM_8.PAS, E-46
 used in COMM_8A.PAS, E-50
 used in COMM_9.PAS, E-56
 used in COMM_9A.PAS, E-60
 used in LAB_5.PAS, K-68
 used in LAB_5A.PAS, K-72
 SEPARATE

SEPARATE_PROG_1.PAS, B-35

SET_TIME
 used in COMM_6.PAS, E-34
 used in NET_1.PAS, F-1
 used in TIME_1.PAS, D-1
 used in TIME_2.PAS, D-7
 used in TIME_2A.PAS, D-15
 used in TIME_3.PAS, D-19

SIGNAL
 used in COMM_2.PAS, E-7
 used in SYNCH_2.PAS, D-32
 used in SYNCH_3.PAS, D-36
 used in SYNCH_4.PAS, D-41

Size attributes
 ATTRIBS_1.PAS, B-20
 TYPECAST_1.PAS, B-23

time from FORTRAN
 TIME_2.PAS, D-7
 TIME_2A.PAS, D-15

time from VMS
 TIME_3.PAS, D-19

time routines
 TIME_1.PAS, D-1

TIME_FIELDS
 used in TIME_1.PAS, D-1

TIME_STRING
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23
 used in COMM_6.PAS, E-34
 used in COMM_6A.PAS, E-36
 used in SYNCH_1.PAS, D-29
 used in SYNCH_6.PAS, D-52
 used in TIME_1.PAS, D-1
 used in TIME_4.PAS, D-25

TIME_VALUE
 used in COMM_6.PAS, E-34
 used in COMM_6A.PAS, E-36
 used in NET_1.PAS, F-1
 used in SYNCH_1.PAS, D-29
 used in SYNCH_2.PAS, D-32
 used in SYNCH_3.PAS, D-36
 used in TIME_1.PAS, D-1
 used in TIME_2.PAS, D-7
 used in TIME_2A.PAS, D-15

TRANSLATE_NAME
 used in COMM_3.PAS, E-14

Typecasting
 ATTRIBS_1.PAS, B-20

typecasting
 TYPECAST_1.PAS, B-23

UNLOCK_MUTEX
 used in SYNCH_5.PAS, D-45
 used in SYNCH_6.PAS, D-52

wait for EVENT
 SYNCH_2.PAS, D-32

wait for objects
 SYNCH_3.PAS, D-36

wait for PROCESS
 SYNCH_4.PAS, D-41

wait routine
 SYNCH_1.PAS, D-29

WAIT_ALL
 used in COMM_2.PAS, E-7
 used in SYNCH_5.PAS, D-45

WAIT_ANY
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23
 used in COMM_5.PAS, E-28
 used in COMM_7.PAS, E-39
 used in COMM_8.PAS, E-46
 used in COMM_8A.PAS, E-50
 used in COMM_9.PAS, E-56
 used in COMM_9A.PAS, E-60
 used in ERRORS_1.PAS, C-10
 used in ERRORS_2.PAS, C-13
 used in LAB_2B.PAS, K-34
 used in LAB_3.PAS, K-47
 used in LAB_4.PAS, K-56
 used in LAB_4A.PAS, K-58
 used in LAB_5.PAS, K-68
 used in LAB_5A.PAS, K-72
 used in LAB_6.PAS, K-78
 used in NET_1.PAS, F-1
 used in SYNCH_1.PAS, D-29
 used in SYNCH_2.PAS, D-32
 used in SYNCH_3.PAS, D-36
 used in SYNCH_4.PAS, D-41
 used in SYNCH_6.PAS, D-52
 used in TIME_3.PAS, D-19

Examples
 communication techniques, E-1
 device drivers, H-1
 exception handling
 EXCEPT_1.PAS, I-1
 file handling
 FILE_1.PAS, G-1
 FILE_2.FOR, G-6
 network facilities
 NET_1.PAS, F-1
 EXCEPT_1.PAS, I-1
 Exception
 kernel response, 15-5
 no user-defined handler present,
 15-5
 stack usage, 15-5
 Exception handler

- possible action, 15-12
- procedure defined, 15-2
- Exception handlers, 15-2
- Exception handling
 - examples
 - EXCEPT_1.PAS, I-1
- EXCEPTION_HANDLER
 - function type
 - used in EXCEPT_1.PAS, I-1
 - predeclared function type, 15-9
- Exceptions
 - asynchronous, 15-2
 - program/programmer response to, 15-2
 - synchronous, 15-2
- Exercise 1
 - solution
 - LAB_1.PAS, K-1
 - LAB_1_C.C, K-14
 - LAB_1_F.FOR, K-7
- Exercise 2
 - solution
 - LAB_2.PAS, K-20
 - LAB_2_C.C, K-20, K-40
 - LAB_2A.PAS, K-27
 - LAB_2B.PAS, K-34
- Exercise 3
 - solution
 - LAB_3.PAS, K-47
 - LAB_3_A.PAS, K-50
- Exercise 4
 - solution
 - LAB_4.PAS, K-56
 - LAB_4A.PAS, K-58
 - LAB_4B.PAS, K-62
- Exercise 5
 - solution
 - LAB_5.PAS, K-68
 - LAB_5A.PAS, K-72
- Exercise 6
 - solution
 - LAB_6.PAS, K-78
 - LAB_6A.PAS, K-81
- Exercises, J-1
 - solutions, K-1
- EXIT
 - EDEBUG command, 7-5
 - predeclared procedure, 3-10, 9-5
- Expedited messages
 - used in COMM_9A.PAS, E-60
- Exponentiation
 - operand restrictions, 2-20
- Exponentiation operator, 2-20
- EXPORT

- default for outer-level
 - declarations, 2-3
- EPASCAL qualifier, 4-2
- example
 - EXPORT_1.PAS, B-10
 - header in module, 2-3
 - header used in MODULE, 8-3
- EXPORT header
 - empty form, 2-4
- EXPORT headers, 2-4
- EXPORT_1.PAS, B-10
- EXPORTed names, 2-3
- Expressions, 2-20
- Expressions and operators, 2-20
- Extension to ISO Pascal
 - %INCLUDE, 2-1
- Extensions to ISO Pascal
 - VAXELN, 2-1
- Extent expression
 - ALIGNED attribute, 2-10
 - BIT attribute, 2-10
 - bound flexible type, 2-7
 - example
 - ATTRIBS_1.PAS, B-20
- Extent expressions
 - CONST declarations, 2-7
 - upper and lower array bounds, 2-7
 - VAR initializers, 2-7
- Extent parameters
 - bound flexible type, 2-7
- EXTERNAL
 - attribute
 - use with VALUE attribute, 2-16
 - directive
 - use with procedures and functions, 2-22
- External files, 12-3
- FAL
 - File Access Listener, 12-7
- FALSE, 2-13
- Fault
 - exception type defined, 15-2
- Faults
 - type of hardware exception, 15-1
- File access
 - remote, 12-7
- File Access Listener
 - EBUILD Network Node
 - characteristics entry, 5-5
 - FAL, 12-7

- File and device specifications, 12-4
- File handling
 - examples
 - FILE_1.PAS, G-1
 - FILE_2.FOR, G-6
- File service
 - memory required, 1-4
- File type
 - default source, 4-1
- File variables
 - as program arguments, 10-5
- FILE\$DENY_DELETE_ACCESS
 - value of predeclared enumerated type FILE\$PROTECTION_TYPES, 12-4
- FILE\$DENY_EXECUTE_ACCESS
 - value of predeclared enumerated type FILE\$PROTECTION_TYPES, 12-4
- FILE\$DENY_READ_ACCESS
 - value of predeclared enumerated type FILE\$PROTECTION_TYPES, 12-4
- FILE\$DENY_WRITE_ACCESS
 - value of predeclared enumerated type FILE\$PROTECTION_TYPES, 12-4
- FILE\$GROUP
 - value of predeclared enumerated type FILE\$PROTECTION_CATEGORIES, 12-4
- FILE\$OWNER
 - value of predeclared enumerated type FILE\$PROTECTION_CATEGORIES, 12-4
- FILE\$PROTECTION_CATEGORIES
 - predeclared enumerated type, 12-4
- FILE\$PROTECTION_TYPES
 - predeclared enumerated type, 12-4
- FILE\$SYSTEM
 - value of predeclared enumerated type FILE\$PROTECTION_CATEGORIES, 12-4
- FILE\$WORLD
 - value of predeclared enumerated type FILE\$PROTECTION_CATEGORIES, 12-4
- FILE_1.PAS, G-1
- FILE_2.FOR, G-6
- FILE_ACCESS
 - predeclared enumerated type, 2-14
- FILE_CARRIAGE_CONTROL
 - predeclared enumerated type, 2-14
- FILE_DISPOSITION
 - predeclared enumerated type, 2-14
- FILE_HISTORY
 - predeclared enumerated type, 2-14
- FILE_RECORD_TYPE
 - predeclared enumerated type, 2-14
- FILE_SHARING
 - predeclared enumerated type, 2-14
- Files
 - access, 12-3
 - direct access, 12-3
 - fixed length records only, 12-3
 - internal and external, 12-3
 - opening and closing, 12-2
 - opening explicitly, 12-2
 - opening implicitly, 12-2
 - record types, 12-3
 - fixed, 12-3
 - variable, 12-3
 - routines, 12-1
 - sequential access, 12-3
 - types, 12-3
 - using, 12-1
 - utility procedures name limits for file specification, 12-4
- Files-11 disk
 - load device at target VAX, 1-4
 - making bootable, 6-1
- FIND
 - predeclared file routine, 12-2
- Flexible type
 - as definition of flexible type, 2-7
- Flexible types, 2-6
- Floppy disk
 - making bootable, 6-2
- FLUSH
 - predeclared file routine, 12-2
- FORTTRAN
 - supported on VAXELN, 0-1
- FORTTRAN Programmer's Guide
 - VAXELN, 0-1

- FP
 - abbreviation for Frame Pointer, 15-5
- FREE
 - state of AREA object, 3-2
- FULL
 - EBUILD qualifier, 5-2
- FUNCTION
 - compilation unit declaration, 8-3
 - declaration in MODULE, 8-3
 - invoked from program block, 2-2
 - routine heading, 8-2
- FUNCTION_BODY
 - reserved word
 - extension to ISO Pascal, 2-2
- FUNCTION_TYPE
 - in function heading, 2-22
 - predeclared procedure
 - used in LAB_1.PAS, K-1
 - used in LAB_2A.PAS, K-27
 - used in LAB_2B.PAS, K-34
 - used in LAB_2.PAS, K-20
- G_FLOATING
 - EPASCAL qualifier, 4-2
- GET
 - predeclared file routine, 12-1
- GET_CONTROL_KEY
 - predeclared text-file routine, 12-2
- GET_TIME
 - predeclared procedure, 9-6
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-14
 - used in SYNCH_1.PAS, D-29
 - used in SYNCH_6.PAS, D-52
 - used in TIME_1.PAS, D-1
- GLOBALDEF
 - attribute of module, 2-3
 - use with EXPORT header, 2-4
- GO
 - EDEBUG command, 7-5
 - restriction on use, 7-4
- Guaranteed image list
 - EBUILD System characteristics
 - entry, 5-4
- HALT
 - EDEBUG command, 7-5
- Handling exceptions, 15-1
- Hardware
 - development system requires, 1-3
 - protecting, 14-1
 - system component, 1-2
 - target system, 1-3
- Hardware exception
 - abort, 15-1
 - fault, 15-1
 - trap, 15-1
- Headers
 - network messages, 11-2
- HELP
 - EDEBUG command, 7-5
- HEX function
 - example
 - RADIX_1.PAS, B-18
- Hexadecimal radix specifier
 - %X, 2-11
- HISTORY\$NEW
 - value of predeclared enumerated
 - type FILE_HISTORY, 2-14
- HISTORY\$OLD
 - value of predeclared enumerated
 - type FILE_HISTORY, 2-14
- HISTORY\$READONLY
 - value of predeclared enumerated
 - type FILE_HISTORY, 2-14
- HISTORY\$UNKNOWN
 - value of predeclared enumerated
 - type FILE_HISTORY, 2-14
- Host System Guide
 - VAXELN, 0-1
- Host VAX load device, 6-4
- HS
 - abbreviation for VAXELN Host
 - System Guide, 0-2
- I/O region size
 - EBUILD System characteristics
 - entry, 5-4
- ICP.SYS
 - Test system from installation, 6-1
- IDENT
 - attribute of module, 2-3
 - example
 - INCLUDE_SOURCE.PAS, B-5
 - SIMPLE.PAS, B-1
- Identifiers
 - avoiding \$ character, 2-1
 - maximum length in VAXELN Pascal, 2-1
 - valid characters, 2-1
 - valid characters in VAXELN
 - Pascal, 2-1
- IF
 - EDEBUG command, 7-5

- IMPORT
 - example
 - IMPORT_1.PAS, B-8
 - header in module, 2-3
 - header used in MODULE, 8-3
- IMPORT_1.PAS, B-8
- %INCLUDE
 - example
 - INCLUDE_SOURCE.PAS, B-5
 - extension to ISO Pascal, 2-1
- INCLUDE
 - EPASCAL qualifier, 4-2
 - example
 - INCLUDE_1.PAS,
 - INCLUDE_1_DEFS.PAS, B-13
 - header in module, 2-3
 - module header
 - EPASCAL command qualifier, 2-4
- INCLUDE headers, 2-4
- INCLUDE_1.PAS, B-13
- INCLUDE_1_DEFS.PAS, B-13
- INCLUDE_SOURCE.PAS, B-5
- Including source text
 - %INCLUDE, 2-1
- Information
 - severity code value 3, 15-4
- Init required
 - EBUILD Program description
 - entry, 5-3
- INIT_TAPE_VOLUME
 - predeclared tape utility, 12-2
- INIT_VOLUME
 - predeclared disk utility, 12-2
 - predeclared procedure
 - used in FILE_1.PAS, G-1
- Initialization
 - aggregate
 - example AGGREG_1.PAS, B-39
- INITIALIZATION_DONE
 - predeclared procedure
 - used in COMM_3.PAS, E-17
 - used in COMM_4.PAS, E-23
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_9.PAS, E-56
 - used in LAB_5.PAS, K-68
 - used in TIME_2A.PAS, D-15
- Initializer, 2-12
 - care using, 2-13
 - for default values in value
 - parameters, 2-12
 - required with EXTERNAL
 - attribute, 2-16
 - required with READONLY
 - attribute, 2-15
 - use in VAR section, 2-12
- Initializing arrays and records, 2-13
- Initializing device, 13-7
- Initializing to binary zero, 2-13
- INLINE
 - EPASCAL qualifier, 4-2
- INSERT_ENTRY
 - predeclared procedure, 2-19, 2-23
 - use in sharing data, 2-19
- INSQUE
 - VAX instruction
 - used by INSERT_ENTRY, 2-19
- Instruction emulation
 - EBUILD System characteristics
 - entry, 5-4
- INTEGER
 - data type
 - use in exponentiation, 2-20
- Inter-job communication, 10-4
- Internal files, 12-3
- Internal time values
 - example
 - TIME_4.PAS, D-25
- Interrupt priority
 - EBUILD Device description entry, 5-3
- Interrupt Priority Levels, 13-4
- Interrupt Service Routine
 - abbreviation ISR, 3-3
 - invoked from program block, 2-2
- Interrupt Service Routine (ISR), 13-2
- Interrupt stack
 - EBUILD System characteristics
 - entry, 5-4
- INTERRUPT_SERVICE
 - declaration in MODULE, 8-3
 - reserved word
 - extension to ISO Pascal, 2-2
- Interrupts
 - definition, 13-2
 - disabling, 13-4
 - unexpected
 - power failure, 13-5
- Intra-job communication, 10-2
- INTRO
 - abbreviation for Introduction
 - to VAXELN, 0-2
- Introduction to VAXELN
 - abbreviation INTRO, 0-2
- IPL

- abbreviation for Interrupt Priority Levels, 13-4
 - hardware range, 13-4
 - software range, 13-4
- ISR
 - abbreviation for Interrupt Service Routine, 13-2
 - DEVICE object, use with, 3-3
 - Interrupt Service Routine, 3-3
 - parameter values, 13-3
- Job
 - definition of, 8-1
 - program execution as, 2-2
 - system component, 1-1
- Job argument data communication example
 - COMM_6.PAS, E-34
 - COMM_6A.PAS, E-36
- Job port, 3-8
 - message limit, 10-8
- Job port message limit
 - EBUILD Program description entry, 5-3
- Job priority
 - EBUILD Program description entry, 5-3
- JOB_PORT
 - predeclared procedure, 3-8, 10-7
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-14, E-17
 - used in COMM_4.PAS, E-23
 - used in LAB_6.PAS, K-78
 - used in TIME_3.PAS, D-19
- KA 630
 - rear panel MicroVAX II, 1-9
- Kb
 - abbreviation for kilobytes, 1-4
- KER\$_BAD_VALUE
 - used in ERRORS_2.PAS, C-13
- KER\$_DISCONNECT
 - used in COMM_9A.PAS, E-60
- KER\$_EXPEDITED
 - used in COMM_9A.PAS, E-60
- KER\$_SUCCESS
 - used in COMM_9A.PAS, E-60
- KER\$CREATE_PROCESS
 - predeclared procedure
 - used in LAB_2_C.C, K-40
- KER\$GB_TIME_SET
 - used in TIME_4.PAS, D-25
- KER\$GQ_IDLE_TIME
 - used in TIME_4.PAS, D-25
- KER\$GQ_PREV_JOB_TIME
 - used in TIME_4.PAS, D-25
- KER\$GQ_START_TIME
 - used in TIME_4.PAS, D-25
- KER\$GQ_SYSTEM_TIME
 - used in TIME_4.PAS, D-25
- KER\$WAIT_ANY
 - predeclared procedure
 - used in LAB_2_C.C, K-40
- KERNEL
 - EBUILD qualifier, 5-2
- Kernel, 1-1
 - debugging, 7-2
 - functions, 1-2
 - memory required, 1-4
 - objects, 3-1
- Kernel mode
 - prerequisite for calling
 - DISABLE_INTERRUPT, 13-4
- Kernel objects, 1-2
 - AREA, 3-1
 - DEVICE, 3-1
 - EVENT, 3-1
 - MESSAGE, 3-1
 - NAME, 3-1
 - PORT, 3-1
 - PROCESS, 3-1
 - SEMAPHORE, 3-1
- Kernel objects and their use, 3-1
- Kernel stack
 - EBUILD Program description entry, 5-3
- LAB_1.PAS, K-1
- LAB_1_C.C, K-14, K-20
- LAB_1_F.FOR, K-7
- LAB_2.PAS, K-20
- LAB_2_C.C, K-40
- LAB_2A.PAS, K-27
- LAB_2B.PAS, K-34
- LAB_3.PAS, K-47
- LAB_3A.PAS, K-50
- LAB_4.PAS, K-56
- LAB_4A.PAS, K-58
- LAB_4A_FUNCDEFS.INC, K-61
- LAB_4B.PAS, K-62
- LAB_5.PAS, K-68
- LAB_5A.PAS, K-72
- LAB_6.PAS, K-78
- LAB_6A.PAS, K-81
- Labels, 2-21
 - implicit declaration, 2-21
- LAN

- Local Area Network, 3-8
- Language run-time
 - memory required, 1-4
- Language supported
 - Programming in, 0-1
- Languages supported by VAXELN, 0-1
- LARGE_INTEGER
 - data type, 2-9
 - operators used with, 2-20
 - suitability for storing time, 9-6
- LARGE_INTEGER use for time values, 2-10
- LEAVE
 - EDEBUG command, 7-5
- LIBRARY
 - EPASCAL qualifier, 4-2
- Licence
 - target system DECnet, 1-5
- LIFO
 - abbreviation for Last In First Off, 15-5
- LINK
 - DCL command, 4-4
 - linker command line, 1-7
 - qualifiers, 4-4
 - stage in development, 1-5
 - VMS DCL command, 1-5
- Linker
 - VMS
 - behaviour with EXTERNAL attribute, 2-16
 - behaviour with VALUE attribute, 2-16
- Linking
 - simple program, 1-7
- LIST
 - EPASCAL qualifier, 4-2
- Literals
 - typecasting not allowed on, 2-18
- LNK\$LIBRARY
 - simplifying the LINK command, 4-4
- LNK\$LIBRARY_1
 - simplifying the LINK command, 4-4
- LOAD
 - EDEBUG command, 7-5
- Load device at target machine, 1-4
- LOAD FILE
 - NCP command, 6-5
- LOCAL
 - Debug option, 7-2
- Local name
 - NAME object, property of, 3-7
- Local volume names, 12-6
- LOCATE
 - predeclared file routine, 12-2
- LOCK_Mutex
 - predeclared procedure, 9-5
 - used in SYNCH_5.PAS, D-45
 - used in SYNCH_6.PAS, D-52
- LOG
 - EBUILD qualifier, 5-2
- Logical links, 11-2
- LOGIN.COM
 - redefining EBUILD, 5-1
 - redefining EPASCAL, 4-1
- LONG
 - attribute
 - use in declarations of variables, 2-15
 - data size attribute, 2-6, 2-10
- LOOP NODE node-id
 - NCP command, 11-9
- Loopback Mirror, 11-8
- LRM
 - abbreviation for VAXELN Pascal Language Reference Manual, 0-2
- LSE
 - abbreviation for Language Sensitive Editor, 1-6
 - text editor, 1-6
- MACHINE_CODE
 - EPASCAL qualifier, 4-2
- Maintenance Operation Monitor
 - MOM, 6-5
- MAP
 - EBUILD qualifier, 5-2
 - EPASCAL qualifier, 4-2
- Map file
 - EBUILD
 - example of brief, 5-7
 - example of full, 5-10
- Master process, 2-2, 8-1
 - initialization of shared data by, 2-20
- MAXINT, 2-13
- Mb
 - abbreviation for megabytes, 1-4
- Mechanism arguments
 - used by exception handlers, 15-9
- Media
 - making bootable, 6-1

- Memory and space
 - development system requires, 1-4
 - target system requires, 1-4
- Memory for language run-time, 1-4
- Memory for VAXELN components, 1-4
- Memory for VAXELN kernel, 1-4
- Memory limit
 - EBUILD System characteristics entry, 5-4
- Memory management and subprocesses, 8-4
- Memory required for
 - device drivers, 1-4
 - file service, 1-4
 - local debugger, 1-4
 - Network service, 1-4
 - remote debugger, 1-4
- MESSAGE
 - contents, 3-6
 - data mapping, 3-6
 - definition, 10-7
 - system data type, 2-9
- Message data
 - mapping, 10-7
- MESSAGE object, 3-1
 - details, 3-5
 - memory allocated, 3-6
 - operations on, 3-6
 - Pool space consumption, 3-6
- Message port
 - NAME object used with, 3-7
- Message ports, 10-7
- Message transmission methods, 10-10
- MESSAGE type
 - parameter to subprocess, 10-3
- MESSAGE utility
 - example
 - ERRORS_2.PAS, C-13
- Messages
 - limit on port, 10-8
- MicroVAX
 - setting default bootstrap selection switch, 6-6
 - target console boot command, 6-5
- MicroVAX II
 - KA 630 rear panel, 1-9
 - language prompt disabling, 6-4
- MicroVMS
 - operating system required for development, 1-5
- Minimum hardware and software, 1-2
- Mixed operands, 2-20
- MOD-PROG-PROC-1.PAS, C-3
- MOD-PROG-PROC-1A.PAS, C-6
- Mode
 - EBUILD Program description entry, 5-3
- Modem
 - EBUILD Terminal descriptions entry, 5-6
- MODULE, 2-3
 - compilation unit declaration, 8-3
 - EPASCAL qualifier, 4-2
 - reserved word, 8-3
 - extension to ISO Pascal, 2-2
- Module headers, 2-3
- Module, program and process example
 - MOD-PROG-PROC-1.PAS, C-3
 - MOD-PROG-PROC-1A.PAS, C-6
- MOM
 - Maintenance Operation Monitor, 6-5
- Monadic operators
 - use with LARGE_INTEGER, 2-20
- Monitoring network events, 6-6
- MOUNT_TAPE_VOLUME
 - predeclared tape utility, 12-2
- MOUNT_VOLUME
 - predeclared disk utility, 12-2
 - predeclared procedure used in FILE_1.PAS, G-1
- MOVE
 - VAX instruction used by READ_REGISTER, WRITE_REGISTER, 2-19
- Multiprocessing, 1-1
- Multitasking, 1-1
 - example, 8-4
- MUTEX
 - example
 - SYNCH_5.PAS, D-45
 - SYNCH_6.PAS, D-52
- Mutex, 9-5
- NAME
 - system data type, 2-9
- Name
 - AREA object, 3-2
 - EBUILD Device description entry, 5-3
- NAME object, 3-1
 - details, 3-7

- local name, 3-7
- naming limit, 3-7
- operations on, 3-7
- pool space consumption, 3-7
- universal name, 3-7
- Name server
 - EBUILD Network Node
 - characteristics entry, 5-5
 - picking, 11-6
 - universal name kept by, 3-7
 - use by NAME object, 3-7
- Name servers, 11-5
- NAME type
 - parameter to subprocess, 10-3
- NAME\$BOTH
 - use with CREATE_NAME procedure, 10-9
 - value of predeclared enumerated type NAME_TABLE, 2-14
- NAME\$LOCAL
 - default for CREATE_NAME call, 10-9
 - use with CREATE_NAME procedure, 10-8
 - value of predeclared enumerated type NAME_TABLE, 2-14
- NAME\$UNIVERSAL
 - use with CREATE_NAME procedure, 10-9
 - value of predeclared enumerated type NAME_TABLE, 2-14
- NAME_TABLE
 - predeclared enumerated type, 2-14
- Named constants
 - predeclared, 2-13
- NCP
 - command CLEAR NODE, 6-7
 - command DEFINE
 - affecting permanent database, 6-4
 - command line, 1-8
 - command LOAD FILE, 6-5
 - command SET
 - affecting volatile database, 6-4
 - command SET LOGGING, 6-6
 - command TRIGGER NODE, 6-6
 - databases, 6-4
 - permanent, 6-4
 - volatile, 6-4
 - DEFINE command
 - privilege required, 6-4
 - Network Control Program, 1-8, 6-2
 - running to check configuration, 1-8
 - SET command
 - privilege required, 6-4
 - SHOW EXECUTOR CHARACTERISTICS, 11-2
- NCP commands supported by VAXELN
 - NML, 11-9
- NET\$OBJECT
 - use in communications, 11-11
- NET_1.PAS, F-1
- Network
 - managing VAXELN, 11-8
- Network Control Program
 - NCP, 6-2
- Network data base
 - Adding target machine, 6-4
- Network device
 - EBUILD Network Node
 - characteristics entry, 5-5
- Network events
 - monitoring, 6-6
- Network facilities, 11-1
 - examples
 - NET_1.PAS, F-1
- Network Management Listener
 - NML, 11-8
- Network segment size
 - EBUILD Network Node
 - characteristics entry, 5-5
- Network Service, 11-1
 - EBUILD Network Node
 - characteristics entry, 5-5
 - implied with remote debug, 11-2
 - port name tables default to local, 10-9
- Network service
 - memory required, 1-4
- Network Services Protocol
 - NSP, 11-1
- NEWBOOT.COM
 - procedure to load bootstrap image file, 6-4
- NML
 - Network Management Listener, 11-8
 - VAXELN, NCP commands supported, 11-9
- Node
 - identification, 11-6
- Node address, 6-3, 11-7
 - area code, 6-3
 - EBUILD Network Node
 - characteristics entry, 5-5
 - node number, 6-3

not required for downline load,
6-5
using, 11-7
Node name, 6-3, 11-7
EBUILD Network Node
characteristics entry, 5-5
not required for downline load,
6-5
using, 11-7
Node number
part of node address, 6-3
use in connecting to VMS, 11-10
Node triggerable
EBUILD Network Node
characteristics entry, 5-5
NODEBUG
qualifier to EDEBUG command,
7-3
Nodename
size in authorization record,
14-6
Non-printing characters in
constants, 2-12
example
EXPORT_1.PAS, B-10
Non-VAXELN Pascal
use of EXTERNAL attribute, 2-16
NONE
Debug option, 7-2
NOTIFY
parameter to CREATE_JOB, 2-5
NOUNDERFLOW
attribute, 2-5
17-NOV-1858 00:00:00.00
base time, 9-6

%O
radix specifier
octal, 2-11
OBJECT
EPASCAL qualifier, 4-2
Object libraries
VAXELN, 4-4
Objectives
student guide, 0-2
Objectname
use in connecting to VMS, 11-10
OCT function
example
RADIX_1.PAS, B-18
Octal radix specifier
%O, 2-11
OPEN
predeclared file routine, 12-1
use in communications, 11-11

OPEN_CIRCUIT
predeclared enumerated type,
2-14
OPER
privilege required for EDEBUG,
6-5
privilege required for NCP
command SET, 6-4
Operands
of different data type, 2-20
Operating system
required for development, 1-5
Operator
typecast ::, 2-17
Operators, 2-20
arithmetic, 2-20
concatenation, 2-21
dyadic, 2-20
exponentiation, 2-20
monadic, 2-20
OPTIMIZE
EPASCAL qualifier, 4-3
Ordinal constants
exporting, 2-4
Ordinal types, 2-5
parameter to subprocess, 10-2
use of attributes with, 2-5
OTHERWISE
clause in CASE statement, 2-22
reserved word
extension to ISO Pascal, 2-2

P0
AREA mapped into, 10-5
region of virtual memory, 8-4
P0 space
use by AREA object, 3-2
used for mapping MESSAGE data,
3-6
P1
region of virtual memory, 8-4
virtual memory
inaccessible process data,
2-18
PACKED ARRAY OF CHAR
string data type, 2-8
PAGE
Predeclared text-file routine,
12-1
Page table slots
EBUILD System characteristics
entry, 5-4
Parameters
CREATE_PROCESS limitations,
10-2

- Parity
 - EBUILD Terminal descriptions
 - entry, 5-6
- Parity type
 - EBUILD Terminal descriptions
 - entry, 5-6
- Pascal
 - supported on VAXELN, 0-1
 - VAXELN extensions to ISO, 2-1
- Pascal Language Reference Manual
 - Parts 1 and 2
 - VAXELN, 0-1
- Pass all
 - EBUILD Console characteristics
 - entry, 5-6
 - EBUILD Terminal descriptions
 - entry, 5-6
- Password
 - size in authorization record,
 - 14-6
- PC
 - abbreviation for Program
 - Counter, 15-5
- Physical memory
 - required for development system,
 - 1-4
- Pointer
 - to bound flexible type
 - use in typecasting, 2-17
 - to type name
 - use in typecasting, 2-17
 - use in sharing data, 2-19
- Pointer data type
 - ANYTYPE allowed, 2-8
- Pointer types, 2-8
 - parameter to subprocess, 10-2
- Polling
 - device access method, 13-4
- Pool size
 - EBUILD System characteristics
 - entry, 5-4
- Pool space consumption
 - AREA object, 3-2
 - DEVICE ISR, 3-3
 - DEVICE object, 3-3
 - EVENT object, 3-5
 - MESSAGE object, 3-6
 - NAME object, 3-7
 - PORT object, 3-8
 - SEMAPHORE object, 3-10
- PORT
 - object for WAIT procedure, 9-3
 - satisfying wait for, 9-3
 - system data type, 2-9
 - use with MESSAGE, 3-6
- Port
 - default message limit, 10-8
 - message limit, 10-8
 - name string, 10-8
- Port name
 - default in NAME\$LOCAL table,
 - 10-9
- PORT object, 3-1
 - details, 3-8
 - maximum message count, 3-8
 - operations on, 3-8
 - pool space consumption, 3-8
 - state, 3-8
- PORT value
 - size, 3-8
 - validity, 3-8
- Port value
 - use with wait procedures, 10-7
- Ports, 10-7
 - EBUILD System characteristics
 - entry, 5-4
 - flow control, 10-13
 - full, 10-13
 - naming, 10-8
 - not connected
 - flow control, 10-13
 - use with messages, 10-7
- POS
 - attribute
 - use on field of packed record,
 - 2-8
- Power failure, 13-5
 - features of recovery ISR, 13-5
- Powerfailure exception
 - EBUILD Program description
 - entry, 5-3
- PREDECESSOR
 - EDEBUG command, 7-5
- Predeclared authorization
 - procedure
 - ELN\$AUTH_ADD_USER, 14-6
 - ELN\$AUTH_MODIFY_USER, 14-6
 - ELN\$AUTH_REMOVE_USER, 14-6
 - ELN\$AUTH_SHOW_USER, 14-6
- Predeclared disk utility
 - DISMOUNT_VOLUME, 12-2
 - INIT_VOLUME, 12-2
 - MOUNT_VOLUME, 12-2
- Predeclared enumerated types,
 - 2-14
 - EVENT_STATE, 2-14
 - FILE\$PROTECTION_CATEGORIES,
 - 12-4
 - FILE\$PROTECTION_TYPES, 12-4
 - FILE_ACCESS, 2-14

- FILE_CARRIAGE_CONTROL, 2-14
- FILE_DISPOSITION, 2-14
- FILE_HISTORY, 2-14
- FILE_RECORD_TYPE, 2-14
- FILE_SHARING, 2-14
- NAME_TABLE, 2-14
- OPEN_CIRCUIT, 2-14
- QUEUE_POSITION, 2-14
- Predeclared file routine
 - CLOSE, 12-1
 - EOF, 12-1
 - FIND, 12-2
 - FLUSH, 12-2
 - GET, 12-1
 - LOCATE, 12-2
 - OPEN, 12-1
 - PUT, 12-1
 - READ, 12-1
 - RESET, 12-1
 - REWRITE, 12-1
 - WRITE, 12-1
- Predeclared file utility
 - COPY_FILE, 12-2
 - CREATE_DIRECTORY, 12-2
 - DELETE_FILE, 12-2
 - DIRECTORY_CLOSE, 12-2
 - DIRECTORY_LIST, 12-2
 - DIRECTORY_OPEN, 12-2
 - PROTECT_FILE, 12-2
 - RENAME_FILE, 12-2
- Predeclared function
 - ADD_INTERLOCKED, 2-19
 - ARGUMENT, 2-16
 - SUBSTR, 2-16
 - TIME_FIELDS, 9-6
 - TIME_STRING, 9-6
 - TIME_VALUE, 9-6
- Predeclared named constants, 2-13
- Predeclared procedure
 - ACCEPT_CIRCUIT, 3-8, 10-11 to 10-12
 - CLEAR_EVENT, 3-5, 9-4
 - CLEAR_SEMAPHORE, 9-4
 - CONNECT_CIRCUIT, 3-8, 10-11 to 10-12
 - CREATE_AREA, 3-2 to 3-3, 10-5
 - CREATE_DEVICE, 3-3 to 3-4, 13-4
 - CREATE_EVENT, 3-5, 9-4
 - CREATE_MESSAGE, 3-6, 10-7
 - CREATE_MUTEX, 9-5
 - CREATE_NAME, 3-7
 - CREATE_PORT, 3-8 to 3-9, 10-7
 - CREATE_PROCESS, 3-9 to 3-10
 - CREATE_SEMAPHORE, 3-11
 - CURRENT_PROCESS, 3-9
 - DELETE, 3-2, 3-4 to 3-7, 3-9 to 3-11, 9-4
 - DELETE_MUTEX, 9-5
 - DISCONNECT_CIRCUIT, 3-8
 - EXIT, 3-10, 9-5
 - GET_TIME, 9-6
 - INSERT_ENTRY, 2-19
 - JOB_PORT, 3-8, 10-7
 - LOCK_MUTEX, 9-5
 - READ_REGISTER, 2-19
 - RECEIVE, 10-9
 - RECEIVE, 10-7, 10-10
 - REMOVE_ENTRY, 2-19
 - RESUME, 3-10
 - SEND, 10-9
 - SET_JOB_PRIORITY, 8-4
 - SET_PROCESS_PRIORITY, 3-10, 8-4
 - SET_TIME, 9-6
 - SIGNAL, 3-2, 3-5, 3-10 to 3-11, 9-5
 - SIGNAL_DEVICE, 3-3
 - START_QUEUE, 2-23
 - SUSPEND, 3-9
 - TRANSLATE_NAME, 3-7
 - UNLOCK_MUTEX, 9-5
 - WAI_ANY, 3-3
 - WAIT_ALL, 3-2 to 3-3, 3-5, 3-9 to 3-11, 9-2
 - WAIT_ANY, 3-2, 3-5, 3-9 to 3-11, 9-2
 - WRITE_REGISTER, 2-19
- Predeclared tape utility
 - DISMOUNT_TAPE_VOLUME, 12-2
 - INIT_TAPE_VOLUME, 12-2
 - MOUNT_TAPE_VOLUME, 12-2
- Predeclared text-file routine
 - EOLN, 12-1
 - GET_CONTROL_KEY, 12-2
 - PAGE, 12-1
 - READLN, 12-1
 - WRITELN, 12-1
- Preparing for loading a system, 1-8
- Priority
 - Job, 8-4
 - Process, 8-4
- PROCEDURE
 - compilation unit declaration, 8-3
 - declaration in MODULE, 8-3
 - invoked from program block, 2-2
 - routine heading, 8-2
- PROCEDURE_BODY
 - reserved word
 - extension to ISO Pascal, 2-2

PROCEDURE_TYPE
 in procedure heading, 2-22
Procedures and functions, 2-22
 declaring, 2-22
 directives, 2-22
PROCESS
 object for WAIT procedure, 9-3
 satisfying wait for, 9-3
 system data type, 2-9
Process
 definition of, 8-1
Process block
 invoked from program block, 2-2
PROCESS object, 3-1
 details, 3-9
 operations on, 3-9
 process states, 3-9
Process priority
 EBUILD Program description
 entry, 5-3
 16 levels of, 3-9
Process states, 8-4
 example showing changes, 8-4
 running, ready, waiting,
 suspended, 3-9
PROCESS type
 parameter to subprocess, 10-3
PROCESS_BLOCK
 compilation unit declaration,
 8-3
 declaration in MODULE, 8-3
 reserved word
 extension to ISO Pascal, 2-2
 routine heading, 8-2
Processes
 data sharing between, 2-18
PROGRAM
 declaration in MODULE, 8-3
 routine heading, 8-2
Program
 arguments
 specifying at system build,
 2-5
 specifying with CREATE_JOB,
 2-5
 arguments to, 2-5
 creating a simple, 1-6
 definition of, 8-1
 execution of, 2-2
 termination message, 2-5
Program arguments
 file variables as, 10-5
PROGRAM block
 compilation unit declaration,
 8-3

Program block, 2-2, 2-5
 allowance per VAXELN Pascal
 program, 2-5
 attribute NOUNDERFLOW, 2-5
 attribute UNDERFLOW, 2-5
 invoking other routines from,
 2-2
 Program development, 4-1
 Program execution, 2-2
 Program structure, 2-2
PROGRAM_ARGUMENT
 function, 10-4
 predeclared function, 2-5
 predeclared procedure
 used in COMM_4.PAS, E-23
 used in COMM_6A.PAS, E-36
PROGRAM_ARGUMENT_COUNT
 function, 10-4
 predeclared function, 2-5
**Programming in VAXELN-supported
 language**
 Course, 0-1
Programs
 DIGITAL supplied, 1-2
 user-written, 1-2
PROTECT_FILE
 predeclared file utility, 12-2
 Proxy access control, 14-2
PSECT
 data, 2-18
 Pseudo variable references, 2-16
PUT
 predeclared file routine, 12-1
QNA
 QBUS Network Adapter, 5-5
QNA-0
 service circuit, 6-4
QUEUE\$CURRENT
 value of predeclared enumerated
 type QUEUE_POSITION, 2-14
QUEUE\$HEAD
 value of predeclared enumerated
 type QUEUE_POSITION, 2-14
QUEUE\$TAIL
 value of predeclared enumerated
 type QUEUE_POSITION, 2-14
QUEUE_POSITION
 predeclared enumerated type,
 2-14
QUEUE_TYPE
 data type, 2-23
 Queues, 2-23

- abbreviation for general register 12, 15-5
 - argument pointer, 15-5
- R13
 - abbreviation for general register 13, 15-5
 - frame pointer, 15-5
- R14
 - abbreviation for general register 14, 15-5
 - stack pointer, 15-5
- R15
 - abbreviation for general register 15, 15-5
 - program counter, 15-5
- Radix specifiers
 - example
 - RADIX_1.PAS, B-18
 - optional apostrophes, 2-11
 - use in improving clarity, 2-11
 - valid characters for constant, 2-12
- RADIX_1.PAS, B-18
- RANDOM.FOR
 - function used by COMM_3, E-20
- READ
 - predeclared file routine, 12-1
- READ_REGISTER
 - predeclared procedure, 2-19
 - use in sharing data, 2-19
- READLN
 - Predeclared text-file routine, 12-1
- READONLY
 - attribute, 2-15
 - use in declarations of variables, 2-15
- READY
 - process state, 8-4
- Ready
 - process state, 3-9
- REAL
 - data type
 - use in exponentiation, 2-20
- REAL type
 - parameter to subprocess, 10-2
- Real-time, 1-1
- RECEIEVE
 - predeclared procedure, 10-9
- RECEIVE
 - predeclared procedure, 10-7, 10-10
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-17
 - used in COMM_4.PAS, E-23
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_8A.PAS, E-50
 - used in COMM_9A.PAS, E-60
 - used in LAB_5.PAS, K-68
 - used in LAB_5A.PAS, K-72
 - used in TIME_3.PAS, D-19
 - provides PORT value, 10-7
- RECEIVE procedure
 - use with MESSAGEs, 3-6
- Record type
 - as definition of flexible type, 2-7
- Record types, 2-8
- RECORD\$FIXED
 - value of predeclared enumerated type FILE_RECORD_TYPE, 2-14
- RECORD\$VARIABLE
 - value of predeclared enumerated type FILE_RECORD_TYPE, 2-14
- Records
 - initialization, 2-13
- Region allocation
 - AREA object, 3-2
- Region mapping
 - AREA object, 3-2
- Register address
 - EBUILD Device description entry, 5-3
- Reloading target machines, 6-6
- REMOTE
 - Debug option, 7-2
- Remote debug
 - implied of Network Service, 11-2
- Remote destinations
 - specifying, 14-3
- Remote file access, 12-7
- REMOVE_ENTRY
 - predeclared procedure, 2-19, 2-23
 - use in sharing data, 2-19
- REMQUE
 - VAX instruction
 - used by REMOVE_ENTRY, 2-19
- RENAME_FILE
 - predeclared file utility, 12-2
- REPLY/ENABLE=NETWORK
 - DCL command, 6-6
- Reserved word
 - FUNCTION_BODY, 2-2
 - INTERRUPT_SERVICE, 2-2
 - MODULE, 2-2

- OTHERWISE, 2-2
- PROCEDURE_BODY, 2-2
- PROCESS_BLOCK, 2-2
- Reserved words
 - extensions to ISO Pascal, 2-2
- RESET
 - predeclared file routine, 12-1
- Resignalling
 - action of exception handler, 15-12
 - effect, 15-12
- RESUME
 - predeclared procedure, 3-10
- REVERT
 - predeclared procedure, 15-14
- REWRITE
 - predeclared file routine, 12-1
- RF
 - abbreviation for VAXELN
 - Run-Time Facilities Guide, 0-2
- ROM
 - Read-Only Memory
 - Downline loader on MicroVAX, 6-4
- Routines, 8-2
 - body, 8-2
 - heading, 8-2
- RTL
 - abbreviation for VMS Run-Time Library Routines Reference Manual, 0-2
- Run
 - EBUILD Program description
 - entry, 5-3
- Run-Time Facilities Guide
 - VAXELN, 0-1
- RUNNING
 - process state, 8-4
- Running
 - process state, 3-9
- RUNOFF, 0-3
- S0
 - region of virtual memory, 8-4
- Scope
 - example
 - SCOPE_1.PAS, B-27
 - SCOPE_2.PAS, B-29
 - SCOPE_3.PAS, B-31
 - SCOPE_3A.PAS, B-33
- SCOPE_1.PAS, B-27
- SCOPE_2.PAS, B-29
- SCOPE_3.PAS, B-31
- SCOPE_3A.PAS, B-33
- SEARCH
 - DCL command used remotely from VMS, 12-8
 - EDEBUG command, 7-5
 - Segment size, 11-2
- SEMAPHORE
 - object for WAIT procedure, 9-3
 - satisfying wait for, 9-3
 - system data type, 2-9
- SEMAPHORE object, 3-1, 3-10
 - maximum count value, 3-10
 - operations on, 3-11
 - pool space consumption, 3-10
- SEMAPHORE type
 - parameter to subprocess, 10-3
- SEND
 - predeclared procedure, 10-9
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-14
 - used in COMM_7A.PAS, E-42
 - used in COMM_8.PAS, E-46
 - used in COMM_8A.PAS, E-50
 - used in COMM_9.PAS, E-56
 - used in COMM_9A.PAS, E-60
 - used in LAB_5.PAS, K-68
 - used in LAB_5A.PAS, K-72
- SEND procedure
 - use with MESSAGES, 3-6
- SEPARATE
 - directive, 8-3
 - use with procedures and functions, 2-22
 - example
 - SEPARATE_PROG_1.PAS, B-35
- Separate routine bodies
 - declaration in MODULE, 8-3
- SEPARATE_PROG_1.PAS, B-35
- Servers
 - DIGITAL supplied programs, 1-2
- Session Control Protocol, 11-1
- SET
 - NCP command requiring privilege, 6-4
- SET BREAK
 - EDEBUG command, 7-5
- SET COMMAND
 - EDEBUG command, 7-6
- SET CONTROL
 - EDEBUG command, 7-6
- SET EXCEPTION BREAK
 - EDEBUG command, 7-6
- SET EXECUTOR NODE
 - NCP command, 11-8
- SET LOG

- EDEBUG command, 7-6
- SET LOGGING
 - NCP command, 6-6
- SET MODE
 - EDEBUG command, 7-6
- SET NODE
 - NCP command, 1-9
- SET PROGRAM
 - EDEBUG command, 7-6
- SET RETURN BREAK
 - EDEBUG command, 7-6
- SET SESSION
 - EDEBUG command, 7-4, 7-6
- SET STEP
 - EDEBUG command, 7-6
- SET TIME
 - EDEBUG command, 7-6
- Set types, 2-6
 - limit on sets of integer, 2-6
 - parameter to subprocess, 10-2
- SET_JOB_PRIORITY
 - predeclared procedure, 8-4
- SET_PROCESS_PRIORITY
 - predeclared procedure, 3-10, 8-4
- SET_TIME
 - predeclared procedure, 9-6
 - used in COMM_6.PAS, E-34
 - used in NET_1.PAS, F-1
 - used in TIME_1.PAS, D-1
 - used in TIME_2.PAS, D-7
 - used in TIME_2A.PAS, D-15
 - used in TIME_3.PAS, D-19
- Severe error
 - severity code value 4, 15-4
- Severity
 - condition value field, 15-4
- SHARE\$NONE
 - value of predeclared enumerated type FILE_SHARING, 2-14
- SHARE\$READONLY
 - value of predeclared enumerated type FILE_SHARING, 2-14
- SHARE\$READWRITE
 - value of predeclared enumerated type FILE_SHARING, 2-14
- Shared data
 - care in modifying, 2-19
 - Initialization of, 2-20
- Sharing data between processes, 2-18
- SHOW
 - EPASCAL qualifier, 4-3
- SHOW BREAK
 - EDEBUG command, 7-6
- SHOW CALLS
 - EDEBUG command, 7-6
- SHOW COMMAND
 - EDEBUG command, 7-6
- SHOW EXECUTOR
 - NCP command, 11-8
- SHOW EXECUTOR CHARACTERISTICS
 - NCP command, 11-2
 - sample display, 11-3
- SHOW JOB
 - EDEBUG command, 7-6
- SHOW KNOWN CIRCUITS
 - NCP command, 11-9
- SHOW KNOWN LINES
 - NCP command, 11-9
- SHOW KNOWN NODES
 - NCP command, 11-7
- SHOW KNOWN OBJECTS
 - NCP command, 11-10
- SHOW MESSAGE
 - EDEBUG command, 7-6
- SHOW MODE
 - EDEBUG command, 7-6
- SHOW MODULE
 - EDEBUG command, 7-6
- SHOW NETWORK
 - DCL command, 11-7
- SHOW NODE
 - NCP command, 1-8
- SHOW NODE node-id
 - NCP command, 11-9
- SHOW node-name CHARACTERISTICS
 - NCP command, 11-2
- SHOW PROCESS
 - EDEBUG command, 7-6
- SHOW PROGRAM
 - EDEBUG command, 7-6
- SHOW SESSION
 - EDEBUG command, 7-7
- SHOW STEP
 - EDEBUG command, 7-7
- SHOW SYMBOL
 - EDEBUG command, 7-7
- SHOW SYSTEM
 - EDEBUG command, 7-7
- SHOW TIME
 - EDEBUG command, 7-7
- SHOW TRANSLATION
 - EDEBUG command, 7-7
- SIGNAL
 - predeclared procedure, 3-2, 3-5, 3-10 to 3-11, 9-5
 - used in COMM_2.PAS, E-7
 - used in SYNCH_2.PAS, D-32
 - used in SYNCH_3.PAS, D-36

- used in SYNCH_4.PAS, D-41
- Signal arguments
 - used by exception handlers, 15-9
- SIGNAL_DEVICE
 - predeclared procedure, 3-3
- SIGNALLED
 - state of AREA object, 3-2
 - state of EVENT object, 3-5
- Simple application, 1-2
 - development, 1-5
- SIMPLE.C
 - example program, 1-6
- SIMPLE.FOR
 - example program, 1-6
- SIMPLE.PAS, B-1
 - example program, 1-6
- Size attributes
 - example
 - ATTRIBS_1.PAS, B-20
 - TYPECAST_1.PAS, B-23
- Software
 - development system requires, 1-5
 - DIGITAL supplied
 - system component, 1-1
 - protecting, 14-1
 - target system requires, 1-5
 - user-supplied
 - system component, 1-1
- Software Product Description
 - abbreviation SPD, 0-2
 - VAXELN, 0-1
- Software requirements, 1-4
- Solution
 - exercise 1
 - LAB_1.PAS, K-1
 - LAB_1_C.C, K-14
 - LAB_1_F.FOR, K-7
 - exercise 2
 - LAB_2.PAS, K-20
 - LAB_2_C.C, K-20, K-40
 - LAB_2A.PAS, K-27
 - LAB_2B.PAS, K-34
 - exercise 3
 - LAB_3.PAS, K-47
 - exercise 3A
 - LAB_3_A.PAS, K-50
 - exercise 4
 - LAB_4.PAS, K-56
 - LAB_4A.PAS, K-58
 - LAB_4B.PAS, K-62
 - exercise 5
 - LAB_5.PAS, K-68
 - LAB_5A.PAS, K-72

- exercise 6
 - LAB_6.PAS, K-78
 - LAB_6A.PAS, K-81
- Solutions to exercises, K-1
- Source text
 - extension to ISO Pascal, 2-1
- SP
 - abbreviation for Stack Pointer, 15-5
- SPD
 - abbreviation for Software Product Description, 0-2
- SPD 28.02.02
 - VAXELN Software Product Description, 0-2
- Speed
 - EBUILD Terminal descriptions entry, 5-6
- SS\$_INTDIV
 - used in EXCEPT_1.PAS, I-1
- Stack, 15-5
 - after exception, 15-11
 - definition, 15-5
 - per-process
 - P1 virtual memory, 2-18
- Stack frame, 15-7
- START_QUEUE
 - predeclared procedure, 2-23
- Statements
 - CASE, 2-21
 - LABEL, 2-21
- Statements in VAXELN Pascal, 2-21
- Status
 - checking, 15-4
 - even values, 15-4
 - odd values, 15-4
- Storage allocation
 - constants and variables, 2-18
- STRING
 - flexible data type, 2-6
 - String data type, 2-8
- String types, 2-7
 - limit on size, 2-7
- Student guide
 - abbreviations used, 0-2
 - conventions used in, 0-2
 - objectives, 0-2
- Subprocess, 2-2, 8-1
- Subprocesses
 - creation after initialization of shared data, 2-20
- SUBSTR
 - predeclared function, 2-16
 - use in pseudo variable references, 2-16

Success
 severity code value 1, 15-4
 SUCCESSOR
 EDEBUG command, 7-7
 SUSPEND
 predeclared procedure, 3-9
 SUSPENDED
 process state, 8-4
 Suspended
 process state, 3-9
 SYNCH_1.PAS, D-29
 SYNCH_2.PAS, D-32
 SYNCH_3.PAS, D-36
 SYNCH_4.PAS, D-41
 SYNCH_5.PAS, D-45
 SYNCH_6.PAS, D-52
 Synchronization
 as provided by kernel, 1-2
 with ports, 10-7
 Synchronization procedures, 9-4
 .SYS
 default file type for system
 image file, 6-5
 .SYS file
 produced by EBUILD, 1-8
 SYS\$NET
 use in communications with VMS,
 11-11
 SYSPRV
 privilege required for NCP
 command DEFINE, 6-4
 SYSTEM
 EBUILD qualifier, 5-2
 System
 building a VAXELN, 1-7
 System build
 for specifying program
 arguments, 2-5
 System builder
 device characteristics
 established with, 3-3
 System components, 1-1
 System data types, 2-9
 System development, 5-1
 System image
 EBUILD System characteristics
 entry, 5-4
 System security, 14-1
 %SYSTEM-F-THIRDPARTY
 error message using EDEBUG, 7-3
 TAPE\$name
 label form for tape volumes,
 12-6
 Target machine

 load device, 1-4
 Target system
 hardware, 1-3
 memory and space required, 1-4
 software required, 1-5
 TELL
 NCP command, 11-8
 Terminal
 EBUILD Terminal descriptions
 entry, 5-6
 Terminal type
 EBUILD Terminal descriptions
 entry, 5-6
 Termination message
 program exit, 2-5
 Time
 specifying absolute form, 9-6
 specifying delta form, 9-6
 Time from FORTRAN
 example
 TIME_2.PAS, D-7
 TIME_2A.PAS, D-15
 Time from VMS
 example
 TIME_3.PAS, D-19
 Time interval
 EBUILD System characteristics
 entry, 5-4
 Time routines
 example
 TIME_1.PAS, D-1
 Time values
 use for LARGE_INTEGER, 2-10
 TIME_1.PAS, D-1
 TIME_2.PAS, D-7
 TIME_2A.PAS, D-15
 TIME_3.PAS, D-19
 TIME_4.PAS, D-25
 TIME_FIELDS
 predeclared function, 9-6
 used in TIME_1.PAS, D-1
 TIME_STRING
 predeclared function, 9-6
 used in COMM_1.PAS, E-1
 used in COMM_2.PAS, E-7
 used in COMM_3.PAS, E-17
 used in COMM_4.PAS, E-23
 used in COMM_6.PAS, E-34
 used in COMM_6A.PAS, E-36
 used in SYNCH_1.PAS, D-29
 used in SYNCH_6.PAS, D-52
 used in TIME_1.PAS, D-1
 used in TIME_4.PAS, D-25
 TIME_VALUE
 predeclared function, 9-6

- used in COMM_6.PAS, E-34
- used in COMM_6A.PAS, E-36
- used in NET_1.PAS, F-1
- used in SYNCH_1.PAS, D-29
- used in SYNCH_2.PAS, D-32
- used in SYNCH_3.PAS, D-36
- used in TIME_1.PAS, D-1
- used in TIME_2.PAS, D-7
- used in TIME_2A.PAS, D-15
- TK50
 - load device at target MicroVAX II, 1-4
- TRANSLATE_NAME
 - network service response, 11-5
 - predeclared procedure, 3-7
 - used in COMM_3.PAS, E-14
 - procedure for getting port value, 10-9
- Trap
 - exception type defined, 15-1
- Traps
 - type of hardware exception, 15-1
- TRIGGER NODE
 - NCP command, 6-6
- TRUE, 2-13
- TU58
 - downline loader storage
 - 11/730 processors, 6-4
 - 11/750 processors, 6-4
 - load device at target VAX, 1-4
 - making bootable, 6-1
- TYPE
 - declaration in MODULE, 8-3
 - EDEBUG command, 7-7
- Type name
 - use in typecasting, 2-17
- Typecast operator ::, 2-17
- Typecast variable references, 2-16
- TYPECAST_1.PAS, B-23
- Typecasting
 - example
 - ATTRIBS_1.PAS, B-20
 - TYPECAST_1.PAS, B-23
 - not allowed on literals, 2-18
 - not allowed on named constants, 2-18
 - not allowed with VALUE
 - attributed variables, 2-16
 - to BYTE_DATA, 2-17
 - to flexible type, 2-17
 - use with ^ANYTYPE, 2-8
 - use with bound flexible type, 2-17
- use with pointer to bound flexible type, 2-17
- use with pointer to type name, 2-17
- use with type name, 2-17
- used in COMM_9A.PAS, E-60
- with BYTE_DATA, 2-10
- UIC
 - System UIC required for NCP DEFINE command, 6-4
 - User identification Code, 3-9
- UNA
 - Unibus Network Adapter, 5-5
- UNA-0
 - service circuit, 6-4
- UNCONNECTED
 - state of PORT object, 3-8
- UNDERFLOW
 - attribute, 2-5
- Universal name
 - memory consumed, 3-7
 - NAME object, property of, 3-7
- Universal name list
 - name server responsibility, 11-5
- Universal names
 - maintained by Network Service, 11-1
- Universal volume names, 12-6
- UNLOAD
 - EDEBUG command, 7-7
- UNLOCK_MUTEX
 - predeclared procedure, 9-5
 - used in SYNCH_5.PAS, D-45
 - used in SYNCH_6.PAS, D-52
- Unwinding, 15-12
- User Identification Code
 - UIC, use with Process object, 3-9
- User stack
 - EBUILD Program description entry, 5-3
- Userdata
 - size in authorization record, 14-6
- Username
 - size in authorization record, 14-6
 - use with Process object, 3-9
- Utilities and Commands, 0-1
- Valid characters
 - use in identifiers, 2-1
- VALIDATE

- EPASCAL qualifier, 4-3
- VALUE
 - attribute
 - use in variable declarations, 2-15
- Value parameters
 - default values
 - initializers used in, 2-12
- VAR
 - declaration in MODULE, 8-3
 - Initializers used in, 2-12
 - use of extent expressions in initializers, 2-7
- VAR parameter
 - non byte-aligned variables, 2-18
 - process block, use in sharing data, 2-19
- Variable references
 - Pseudo, 2-16
- Variables, 2-15
 - addresses of, 2-18
 - storage allocation, 2-18
- VARYING_STRING
 - flexible data type, 2-6
 - String data type, 2-8
- VAX instruction
 - ADAWI
 - used by ADD_INTERLOCKED, 2-19
 - INSQUE
 - used by INSERT_ENTRY, 2-19
 - used by INSERTENTRY, 2-23
 - MOVE
 - used by READ_REGISTER, 2-19
 - used by WRITE_REGISTER, 2-19
 - REMQUE
 - used by REMOVE_ENTRY, 2-19
 - used by REMOVEENTRY, 2-23
- VAX Language-Sensitive Editor
 - VAXELN Pascal Guide, 0-1
- VAXELN
 - capabilities and applications, 1-1
- VAXELN Application Design Guide
 - abbreviation ADG, 0-2
- VAXELN extensions to ISO Pascal, 2-1
- VAXELN Host System Guide
 - abbreviation HS, 0-2
- VAXELN Pascal
 - statements in, 2-21
- VAXELN Pascal Language Reference Manual
 - abbreviation LRM, 0-2
- VAXELN Run-Time Facilities Guide
 - abbreviation RF, 0-2
- Vector address
 - EBUILD Device description entry, 5-3
- Version
 - Software Version 2.3, 0-1
- Virtual memory, 8-4
 - Pl space, 2-18
- Virtual page file quota
 - required for development system, 1-4
- Virtual size
 - EBUILD System characteristics entry, 5-4
- VMS
 - network connections to and from, 11-10
 - operating system
 - required for development, 1-5
- VMS linker
 - behaviour with EXTERNAL attribute, 2-16
 - behaviour with VALUE attribute, 2-16
 - global values and EXPORT header, 2-4
- VMS Run-Time Library Routines
 - Reference Manual
 - abbreviation RTL, 0-2
- Volume
 - default, 12-5
- Volume labels, 12-5
- Volume names
 - specifying, 12-5
- WAIT
 - synchronization procedures, 9-2
- Wait for EVENT
 - example
 - SYNCH_2.PAS, D-32
- Wait for objects
 - example
 - SYNCH_3.PAS, D-36
- Wait for PROCESS
 - example
 - SYNCH_4.PAS, D-41
- Wait routine
 - example
 - SYNCH_1.PAS, D-29
- WAIT_ALL
 - predeclared procedure, 3-2 to 3-3, 3-5, 3-9 to 3-11, 9-2
 - used in COMM_2.PAS, E-7
 - used in SYNCH_5.PAS, D-45
- WAIT_ANY

- predeclared procedure, 3-2 to
 - 3-3, 3-5, 3-9 to 3-11, 9-2
 - used in COMM_1.PAS, E-1
 - used in COMM_2.PAS, E-7
 - used in COMM_3.PAS, E-17
 - used in COMM_4.PAS, E-23
 - used in COMM_5.PAS, E-28
 - used in COMM_7.PAS, E-39
 - used in COMM_8.PAS, E-46
 - used in COMM_8A.PAS, E-50
 - used in COMM_9.PAS, E-56
 - used in COMM_9A.PAS, E-60
 - used in ERRORS_1.PAS, C-10
 - used in ERRORS_2.PAS, C-13
 - used in LAB_2B.PAS, K-34
 - used in LAB_3.PAS, K-47
 - used in LAB_4.PAS, K-56
 - used in LAB_4A.PAS, K-58
 - used in LAB_5.PAS, K-68
 - used in LAB_5A.PAS, K-72
 - used in LAB_6.PAS, K-78
 - used in NET_1.PAS, F-1
 - used in SYNCH_1.PAS, D-29
 - used in SYNCH_2.PAS, D-32
 - used in SYNCH_3.PAS, D-36
 - used in SYNCH_4.PAS, D-41
 - used in SYNCH_6.PAS, D-52
 - used in TIME_3.PAS, D-19
- WAITING
 - process state, 8-4
- Waiting
 - process state, 3-9
- Warning
 - severity code value 0, 15-4
- WARNINGS
 - EPASCAL qualifier, 4-3
- WORD
 - attribute
 - use in declarations of
 - variables, 2-15
 - data size attribute, 2-6, 2-10
- Working set
 - required for development system,
 - 1-4
- WRITE
 - predeclared file routine, 12-1
- WRITE_REGISTER
 - predeclared procedure, 2-19
 - use in sharing data, 2-19
- WRITELN
 - Predeclared text-file routine,
 - 12-1
- %X
 - radix specifier
 - hexadecimal, 2-11
- ZERO
 - function, 2-13
 - use as initializer, 2-13
- ZERO EXECUTOR
 - NCP command, 11-9
- ZERO KNOWN CIRCUITS
 - NCP command, 11-9
- ZERO KNOWN LINES
 - NCP command, 11-9
- ZERO NODE node-id
 - NCP command, 11-9