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

Printed in U.K.

CONTENTS

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:

- Ada *
- C
- FORTRAN
- Pascal

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

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

- 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]

- Development system:

  (a) MicroVAX I

  (b) MicroVAX II

  (c) any 700 series VAX

  (d) any 8000 series VAX

- 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

- 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

- 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

- 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

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
```

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 _

- 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)

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:

- 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

- 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

- 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.

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

2.7 PREDECLARED ENUMERATED TYPES

[LRM:4-14]

Enumerated Type	Values
EVENT_STATE	EVENT$CLEARED, EVENT$SIGNALED
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:

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

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

2.8.1 READONLY Attribute

[LRM:5-3, Appendix A]

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

2.8.2 VALUE Attribute

[LRM:5-4, Appendix A]

- provides information to the VMS linker about data item

- 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;
```

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.

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.

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:

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

3.1.1 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 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

- 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
              );
```

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,
```

```
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

- 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"
```

The full list of qualifiers to EPASCAL is:

Qualifier	Negative form	Default
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

Qualifier	Negative form	Default
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

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:

- ELN$:RTLSHARE.OLB
- ELN$: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"
$ !
```

The full list of EBUILD qualifiers is:

Qualifier	Negative form	Default
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

Entry	Responses	Default(s)
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

Entry	Responses	Default(s)
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

5.1.3 System Characteristics Menu

Entry	Responses	Default(s)
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

Entry	Responses	Default(s)
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

5.1.5 Terminal Descriptions Menu

Entry	Responses	Default(s)
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

Entry	Responses	Default(s)
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
```

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

```
------------------------- EBUILD COMMAND ------------------

        $ EBUILD /NOEDIT /MAP /FULL SIMPLE

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

VAXELN System Builder                                27-MAY-1987 12:30:19.34
ELN V2.3-00                                          27-MAY-1987 12:30:19.34

System file
-----------
SIMPLE                  DISK$INSTRUCT:[SHONE.VAXELN]SIMPLE.SYS;2

Kernel
------
KERNEL                  SYS$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                SYS$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              SYS$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 = %O774440
                       Vector = %O120
                       Priority = 4
                       BI number = 0
                       Adapter number = 0

Terminals
---------

Shareable images
----------------
NETWORK                SYS$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
```

```
                           Noshr Write   80012000        2
                           Fixup vector  80012400        1

PASCALMSC                SYS$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                      SYS$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                SYS$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                SYS$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                SYS$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                  SYS$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

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:

- node address
- 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.

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
```

<u>Circuits and Lines - MicroVAX</u>

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 EDEBUG thus:

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

Using EDEBUG 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:

- 11/730 - >>> B XE0 (DEUNA loader)

- 11/750 - >>> B DDA0 (Console storage)

- 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 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:

- 11/730 - setting is performed by the NEWBOOT procedure
- 11/750 - set default boot device switch to "A"
- 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
```

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:

- debug access to one or more VAXELN target systems at the same time

- availability of the debug symbol table provided by compilers, giving access to variable names, labels and source-line information

- 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:

- allows you to debug your system from your target VAX

- those facilities available when debugging remotely are unavailable when debugging locally

- a network connection between host and target is not essential.

- 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:

- NONE - a fully debugged system not requiring the presence of debugging facilities

- LOCAL - the local debugger is built into the system image

- REMOTE - the remote debugger is built into the system and EDEBUG is used to access the target remotely over an Ethernet

- 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
```

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 addressexpression

- 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.

- SET BREAK /qualifier address-expression DO (one-line-command) - sets breakpoint at specified address. Qualifiers are /JOB, /ALL, /KERNEL

- SET COMMAND identifier DO (one-line-command) - create a command

- SET CONTROL - opposite of CANCEL CONTROL

- 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

- SET LOG file-specification - causes logging of debug session. Omission of file-specification stops logging

- 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.

- SET PROGRAM image-file-specification - change session's program image

- SET RETURN BREAK - stop session when current routine returns

- SET SESSION /qualifier process-specifier nodename - change session to another, optionally on another node. Qualifiers are /GO and /KERNEL

- SET STEP - change default stepping action. Options are INTO LINE, OVER LINE, INTO INSTRUCTION, OVER INSTRUCTION, INTO SOURCE, OVER SOURCE

- SET TIME time-string nodename - set the system time on specified node using VMS time notation

- SHOW BREAK - display breakpoint information

- SHOW CALLS - display call history

- SHOW COMMAND identifier - display one or all commands established with SET COMMAND. Optional qualifier /ALL

- SHOW JOB job-id nodename - display information about all processes in a job. Job-id may be string, integer or identifier

- SHOW MESSAGE expression - show text for value of an exit status

- SHOW MODE - show current operating modes for the debugger

- SHOW MODULE provide information about the program of the current session

- SHOW PROCESS process-specifier nodename - display system state of a job or process, optionally on another node. Qualifier /ALL

- SHOW PROGRAM name - show system information about installed program. Qualifier /ALL

- SHOW SESSION process-specifier nodename - show debug state of one or all debugging sessions, optionally on another node. Qualifier /ALL

- SHOW SYMBOL pathname identifier - displays information about a symbol. Qualifier /DEFINE

- SHOW SYSTEM nodename - displays memory, CPU time and jobs on system, optionally on another node

- SHOW TIME nodename - shows current system time, optionally on another node

- SHOW TRANSLATION identfier OR string nodename - displays translation of a PORT object value, optionally on another node

- STEP - execute next line or instruction. Also /INTO /LINE, /OVER /LINE, /INTO /INSTRUCTION, /OVER /INSTRUCTION, /INTO /SOURCE, /OVER /SOURCE

- SUCCESSOR expression - move scope of current session's reference a number of call frames forwards in the calling order

- TYPE module-name expression:expression - display source lines in range specified

- 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 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.

8.4 ROUTINES IN VAXELN

A routine has two components:

- a heading and
- 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:

- PROGRAM
- FUNCTION
- PROCEDURE
- PROCESS_BLOCK

8.4.2 Body Of Routines

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

- a series of declarations - constants, types, variables and other routines
- 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

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:

	Highest	Lowest	Default
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

```
               +----------------------+
0000 0200:     |   Read/write data    |
               |                      |
               |----------------------|
               | Read-only code/data  |
               |                      |
               |----------------------|
   P0          |      Heap data       |
               |                      |
               |                      |
               |     Message data     |
               |                      |
               |----------------------|
3FFF FFFF:     |        Unused        |
               +----------------------+
4000 0000:     |                      |
               |                      |
               |                      |
               |                      |
               |                      |
               |   User mode stack    |
   P1          |   (if necessary)     |
               |----------------------|
               |  Kernel mode stack   |
               |  (at least 2 pages)  |
               |----------------------|
               | Debug context block  |
7FFF FFFF:     |     (if needed)      |
               +----------------------+
8000 0000:     |     Kernel image     |
               |----------------------|
               |   Program 1 image    |
               |          .           |
               |          .           |
               |                      |
   S0          |   Program n image    |
               |----------------------|
               |    Run-time images   |
               |----------------------|
               | Kernel pool and data |
               |----------------------|
BFFF FFFF:     |        Unused        |
               +----------------------+
```

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:

- making things happen together
- preventing things happening together
- making things happen in order
- 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.

9.1.3 Making Things Happen In Order

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

- process A feeds empty bottles onto the line
- process B controls emission of liquid into empty bottles
- process C caps the filled bottles
- process D labels the filled bottles
- 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

The are two wait routines in VAXELN:

- WAIT_ANY
- 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 satisifed 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 satisifed 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)

- 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:

- CREATE_EVENT
- CLEAR_EVENT
- CREATE_SEMAPHORE
- DELETE
- EXIT
- 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:

- CREATE_MUTEX
- DELETE_MUTEX
- LOCK_MUTEX
- UNLOCK_MUTEX

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:

- communication within one VAXELN processor - single node
- 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.

10.2 COMMUNICATION WITHIN A SINGLE JOB ON A SINGLE VAXELN NODE

```
                 +------- NODE HENRY -------+
                 |                          |
                 |        +---------+       |
                 |        |         |       |
                 |        | JOB  A  |       |
                 |        |         |       |
                 |        +---------+       |
                 |                          |
                 +-------------+------------+
                               |
                               |                      E T H E R N E T
-------------------------------+---------------------------------------
```

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.

10.3 COMMUNICATION BETWEEN MULTIPLE JOBS ON A SINGLE VAXELN NODE

```
           +------- NODE  WATT -------+
           |   +-----+                |
           |   |job A|    +-----+     |
           |   +-----+    |job B|     |
           |              +-----+     |
           |       +-----+            |
           |       |job C|            |
           |       +-----+            |
           +-------------+------------+
                         |
                         |                   E T H E R N E T
-------------------------+----------------------------------------
```

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.

<u>Position Independent Area</u>

```
            +---------------+                     +---------------+
0000 0000:  | P0 in job A   |                     |  P0 in job B  |
            |               |    +----------+     |               |
            |               |    |  AREA    |     |               |
            |               |    |          |     |               |
            |_______________|    |          |     |               |
            | mapping in  A |    +----------+     |               |
            |---------------|                     |               |
            |               |                     |               |
            |               |                     |_______________|
            |               |                     | mapping in  B |
            |               |                     |---------------|
3FFF FFFF:  |               |                     |               |
            +---------------+                     +---------------+
```

<u>Position Dependent Area</u>

```
            +---------------+                     +---------------+
0000 0000:  | P0 in job A   |                     |  P0 in job B  |
            |               |    +----------+     |               |
            |               |    |  AREA    |     |               |
            |               |    |          |     |               |
            |_______________|    |          |     |_______________|
            | mapping in  A |    +----------+     | mapping in  B |
            |---------------|                     |---------------|
            |               |                     |               |
            |               |                     |               |
            |               |                     |               |
            |               |                     |               |
3FFF FFFF:  |               |                     |               |
            +---------------+                     +---------------+
```

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

  - 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:

- NAME$LOCAL
- NAME$UNIVERSAL
- 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:

- SEND
- 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:

- 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.
- REPLY - provides a port value for replies. Default is sender's job port.
- 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.

```
                port
 +----------+  /
 | JOB R    | / +--------<----------------------[EXPEDITED]
 |          |/  |                            (Max. 16 bytes)
 |          =   |
 |          =   v [message] [message] [message] [message] [message] ....
 |          =
 |          |
 +----------+
```

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:

```
                 port                        port
+----------+  /                                \  +----------+
| JOB X    | /                                  \ | JOB Y    |
|          |/                                    \|          |
|          =                                      =          |
|          =---------- C I R C U I T ----------=          |
|          =                                      =          |
|          |                                      |          |
+----------+                                      +----------+
```

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.

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:

- DESTINATION_PORT - destination port value for the connection request message. This may be omitted if a destination name is supplied.

- 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

- FULL_ERROR - TRUE or FALSE. The default of FALSE implies that this process will wait on a SEND if the other port is full

- CONNECT_DATA - a VARYING_STRING of 16 bytes that contains the connector's message to X - "Hello I am Y"

- 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:

- the sender process enters the waiting state or

- 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:

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

```
          +------- NODE HENRY -------+    +------- NODE JOULE -------+
          |   +-----+                |    |      +-----+             |
          |   |job A|     +-----+    |    |      |job 1|  +-----+    |
          |   +-----+     |job B|    |    |      +-----+  |job 2|    |
          |               +-----+    |    |   +-----+     +-----+    |
          |        +-----+           |    |   |job 3|  +-----+       |
          |        |job C|           |    |   +-----+  |job 4|       |
          |        +-----+           |    |            +-----+       |
          +-------------+------------+    +-------------+------------+
                        |                               |
                        |                               |   E T H E R N E T
----------------------+--------------------------------+------------------
```

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

End nodes and routing nodes

```
                                                            The world....
                                                                  ^
                 END-NODES                                        |
                /    |    \                  +----------+   +----+----+
               /     |     \                 | R S X    +---+ V M S   | <-router
              /      |      \                +----------+   +----+----+
             /       V       \                                   |
+----------+  +----------+  +----------+     +----------+   +----+----+
| VAXELN   |  | VAXELN   |  | VAXELN   |     | R S T S  +---+ V M S   | <-router
+----+-----+  +----+-----+  +----+-----+     +----------+   +----+----+
     |             |             |                               |      Ethernet
-----+-------------+-------------+-------------------------------+-------------
```

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
```

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.

<u>Name service requests sent to name server...</u>

```
                                                       Ethernet->||
+------- JOB A -------+                                          ||
|                     |                                          ||
| TRANSLATE_NAME (... |   +--------+   +---------+               ||
|                     |   |        |   | Network |  to NAME SERVER ||
| CREATE_NAME    (... |-->| KERNEL |-->|         |------------------>||
|                     |   |        |   | service |               ||
| DELETE_NAME    (... |   +--------+   +---------+               ||
|                     |                                          ||
+---------------------+                                          ||
```

<u>...name server returns status to process</u>

```
                                                                 ||
+------- JOB A -------+                                          ||
|                     |                                          ||
| STATUS := ... );    |   +--------+   +---------+               ||
|                     |   |        |   | Network | from NAME SERVER ||
| STATUS := ... );    |<--| KERNEL |<--|         |<------------------||
|                     |   |        |   | service |               ||
| STATUS := ... );    |   +--------+   +---------+               ||
|                     |                                          ||
+---------------------+                                Ethernet->||
```

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

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:

- 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

- SHOW EXECUTOR parameters

  The parameters are:

  SUMMARY, STATUS, CHARACTERISTICS or COUNTERS

- SHOW KNOWN CIRCUITS parameters

  The parameters are:

  SUMMARY, COUNTERS

- SHOW KNOWN LINES parameters

  The parameters are:

  SUMMARY, COUNTERS

- SHOW NODE node-id parameters

  The parameters are:

  SUMMARY, COUNTERS

- ZERO EXECUTOR

- ZERO KNOWN CIRCUITS

- ZERO KNOWN LINES

- 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:

```
$ 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

- o CLOSE
- o FIND
- o LOCATE
- o FLUSH

and for text files.

- o GET_CONTROL_KEY

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

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

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

- o MOUNT_VOLUME
- o INIT_VOLUME
- o DISMOUNT_VOLUME

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

- o MOUNT_TAPE_VOLUME
- o INIT_TAPE_VOLUME
- o 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) - <u>but records must be FIXED length</u>

12.6 RECORD TYPES

Two types of record are supported:

- o fixed length
- o variable length

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       "DUA1 ERRLOG", "DUA2 DATA"

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

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

```
characteristic /volumes=("DUA1 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:

- 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 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 "DUA1 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:

> "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      DHVDRIVER.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:

```
                                  +------------------+
                                  | DEVICE DRIVER    |
+--------+                        |                  |
|        |                        |     +-----+      |
| DEVICE +->INTERRUPT--> KERNEL->-+---->| ISR |      |
|        |                        |     +--+--+      |
+--------+                        |        ^         |
                                  |        |         |
                                  +--------+---------+
                                           |
                                           v
+---------+                       +------------------+
| PROGRAM |<--------------------->| COMMUNICATION    |
+---------+                       |      REGION      |
                                  +------------------+
```

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

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:

- 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:

- AXV_INITIALIZE - for ADV or AXV analogue to digital converters
- KWV_INITIALIZE - for KWV programmable realtime clocks
- DLV_INITIALIZE - for DLV asynchronous serial line controllers
- 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.

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:

- control operating modes - kernel mode is freely available
- limit priorities - programs can change priorities freely
- 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:

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

The diagram below shows how the two access methods work

```
      Node "MAYHEM"                                   Node "BEDLAM"
     +---------------+                               +---------------+
     | Application   |<-- User: BRAINS               | Application   |
     +---------------+                               +---------------+
     |   Network &   |                               |   Network &   |
     | Authorization |                               | Authorization |
     |    Services   |                               |    Services   |
     +-------+-------+                               +-------+-------+
             |                                               ^
             v                                               |
             +--------------- "BRAINS,MAYHEM" ---------------+

                              Proxy Access Control
```

```
      Node "MAYHEM"                                   Node "BEDLAM"
     +---------------+                               +---------------+
     | Application   |                               | Application   |
     +---------------+                               +---------------+
     |   Network &   |                               |   Network &   |
     | Authorization |<-- User: HAL2001              | Authorization |
     |    Services   |    Password: ARTHURC          |    Services   |
     +-------+-------+                               +-------+-------+
             ^                                               |
             |                                               v
             +------------- "HAL2001 ARTHURC" ---------------+

                           Destination Authorization
```

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'

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

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

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

14.5.2 Connecting To VAXELN Nodes

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

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

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

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

14.6 INCLUDING AUTHORIZATION IN A SYSTEM

The relevant network node characterstics 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

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".

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 reponsible 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:

```
 31     28 27                                              3 2          0
+--------+-------------------------------------------------+----------+
| cntrl  |             condition identification            | severity |
+--------+-------------------------------------------------+----------+

         |                                                 |
         v                                                 v
         |                                                 |
         v                                                 v
          27                     16 15                     3
         +------------------------+------------------------+
         |    facility number     |     message number     |
         +------------------------+------------------------+
```

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.

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:

```
+----------------------------------------+
|                                        | :(SP)
|                                        |
|               Procedure                |
|                                        |
|                 Local                  |
|                                        |
|                Storage                 |
|                                        |
|                                        |
|                                        |
+----------------------------------------+
|   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 R10               | :(FP) + 60
+----------------------------------------+
|                saved R11               | :(FP) + 64
+----------------------------------------+
```

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

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

```
+---------------------------------------+
| Procedure Local Storage               | :(SP)
+---------------------------------------+
| Active Call Frame                   E | :(FP)
+---------------------------------------+
| Procedure Local Storage               |
+---------------------------------------+
| Previous Call Frame                 D |     (P1)
+---------------------------------------+    higher
| Procedure Local Storage               |   addresses
+---------------------------------------+       |
| Previous Call Frame                 C |       |
+---------------------------------------+       |
| Procedure Local Storage               |       v
+---------------------------------------+
| Previous Call Frame                 B |
+---------------------------------------+
| Procedure Local Storage               |
+---------------------------------------+
| Previous Call Frame                 A |
+---------------------------------------+
```

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.

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:

```
              Call frame(s) for
             exception handler(s)
                                                     ^
        |                           |                |
        |                           |                |
        +-----------------------+---+      Stack growing |
        |                       | 2 |      to lower addresses
        +-----------------------+---+
        |        Address of     ----+----------+
        |       Signal array        |          |
        +---------------------------+          |
        |        Address of     ----+-----+    |
        |     Mechanism array       |     |    |
        +-----------------------+---+     |    |
        |                       | 4 |<----+    |
        +-----------------------+---+          |
        |           Frame           |          |
        +---------------------------+          |
        |           Depth           |          |
        +---------------------------+          |
        |         Saved R0          |          |
        +---------------------------+          |
        |         Saved R1          |          |
        +---------------------------+          |
        |     Signal/stop code      |          |
        +-----------------------+---+          |
        |                       | N |<---------+
        +-----------------------+---+
        |     Exception value       |
        +---------------------------+
        |   Additional exception    |
        |        arguments          |
        +---------------------------+
        |      Exception PC         |
        +---------------------------+
        |      Exception PSL        |
        +---------------------------+
        |                           |<---- Value of SP
                                           before exception
```

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 - RESIGNALLING

If the exception handler decides that it is unable to handle a particular condition it will return a BOOLEAN value FALSE. This is termed RESIGNALLING (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 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

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

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

```
{-----------------------------------------------------------------
SOURCE:         SIMPLE.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal

COMPILE:        $ EPASCAL /LIST /DEBUG SIMPLE

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

BUILD:          $ EBUILD /NOEDIT SIMPLE

NOTES:          1) Command procedure SIMPLE.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
-----------------------------------------------------------------}
```

```

MODULE Simple [IDENT ('V1.000')];

{ The IDENT field appears in the object file's MODULE HEADER with the
  module's name thus:

        1.  MODULE HEADER (OBJ$C_HDR_MHD), 53 bytes

                structure level: 0
                maximum record size: 512
                module name: "SIMPLE"
                module version: "V1.000"
                creation   date/time: 13-FEB-1987 11:21

        _______________________________________________

        To obtain this information proceed thus:
                $ EPASCAL SIMPLE
                $ ANALYZE /OBJECT SIMPLE
}
PROGRAM Simple (OUTPUT);

    BEGIN

        WRITELN (  'A simple program to test VAXELN' );

END { of PROGRAM Simple }.
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

```
$       ! SIMPLE.COM
$       ! Command procedure to build the VAXELN module SIMPLE
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile_simple
$ !
$       SET VERIFY
$         EPASCAL /LIST SIMPLE
$         LINK SIMPLE
$       GOTO Switch_off_verify
$ !
$ Compile_Simple:
$               @ELN_COMPILE_1  SIMPLE
$ Link_Simple:
$               @ELN_LINK_1     SIMPLE
$ System_build:
$               @ELN_EBUILD_1   SIMPLE
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
program SIMPLE /debug
```

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.

```
$       ! ELN_COMPILE_1.COM
$       ! Command procedure to compile a VAXELN Pascal program
$       !
$ Compile:
$       SET VERIFY
$       EPASCAL -
                /LIST -
                /DEBUG 'P1'
$       SET NOVERIFY
$       EXIT
```

```
$       ! ELN_LINK_1.COM
$       ! Command procedure to link VAXELN object modules
$       !
$ Link:
$       SET VERIFY
$       LINK -
                /DEBUG 'P1' -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$       SET NOVERIFY
$       EXIT
```

```

$       ! ELN_EBUILD_1.COM
$       ! Command procedure to build a VAXELN system from existing
$       ! 'P1'.DAT file
$       !
$ System_build:
$       SET VERIFY
$       EBUILD -
                /NOEDIT 'P1'
$       SET NOVERIFY
$       EXIT
```

B.3 EXAMPLE OF %INCLUDE - INCLUDE_SOURCE.PAS

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

```
{-----------------------------------------------------------------------
SOURCE:          INCLUDE_SOURCE.INC

PURPOSE:         For inclusion in INCLUDE_SOURCE.PAS

USED BY:         INCLUDE_SOURCE.PAS
----------------------------------------------------------------------}
PROCEDURE Show_scope;

BEGIN
  WRITELN ( 'This is from procedure SHOW_SCOPE' );
  WRITELN ( ' about to write GREETING.........' );
  WRITELN ( Greeting );

END { PROCEDURE Show_scope };
```

The program looks like this:

```
{-----------------------------------------------------------------------
SOURCE:         INCLUDE_SOURCE.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal with outer level declaration of a CONST

NEEDS:          1) INCLUDE_SOURCE.INC for the %INCLUDE

COMPILE:        $ EPASCAL /LIST /DEBUG INCLUDE_SOURCE

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

BUILD:          $ EBUILD /NOEDIT INCLUDE_SOURCE

NOTES:          1) Command procedure INCLUDE_SOURCE.COM compiles,
                   links and builds this module into a VAXELN system
                   or for running under VMS
-----------------------------------------------------------------------}

MODULE INCLUDE_SOURCE [IDENT ('V1.000')];

        CONST   { This outer level declaration is EXPORTed explicitly }
                Greeting = 'A simple program to test VAXELN';

PROGRAM Simple (OUTPUT);

  CONST
        Greeting = 'This is an inner declaration of the same name';

{ Code for the procedure SHOW_SCOPE }
        %INCLUDE 'INCLUDE_SOURCE.INC/LIST'

    BEGIN

        WRITELN (  Greeting );
        { prints "This is an inner declaration of the same name" }

        Show_scope;

    END { of PROGRAM }.
END     { of MODULE  };
```

B.3.1 Running INCLUDE_SOURCE

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

```
$       ! INCLUDE_SOURCE.COM
$       ! Command procedure to build the VAXELN
$       ! module INCLUDE_SOURCE
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST INCLUDE_SOURCE
$         LINK          INCLUDE_SOURCE
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  INCLUDE_SOURCE
$ Link:
$               @ELN_LINK_1     INCLUDE_SOURCE
$ System_build:
$               @ELN_EBUILD_1   INCLUDE_SOURCE
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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
```

B.4 EXAMPLE OF IMPORT AND EXPORT - IMPORT_1.PAS

```
{-----------------------------------------------------------------------
SOURCE:          IMPORT_1.PAS

PURPOSE:         This module converts a temperature in CELSIUS to
                 FAHRENHEIT for user input from a terminal.
                 It demonstrates the IMPORT statement in VAXELN
                 Pascal

COMPILE:         $ EPASCAL /DEBUG IMPORT_1, -
                 _$ EXPORT_1/MODULE

LINK:            $ LINK    /DEBUG IMPORT_1, EXPORT_1 -
                 _$ ,ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT IMPORT_1

NOTES:           1) Command procedure IMPORT_1.COM compiles, links and
                    builds this module into a VAXELN system or for
                    running under VMS
-----------------------------------------------------------------------}
MODULE Temperature_converter [IDENT ('V1.000')];

IMPORT  Clear_screen,
        Prompt,
        Celsius_input,
        Fahrenheit_value,
        Conversion_factor,
        Fahrenheit_zero;

PROGRAM Convert_C_to_F ( INPUT, OUTPUT );

  VAR
        Celsius,
        Fahrenheit      : REAL;

{ ------------------------------------------------------------------- }

PROCEDURE Describe_program;

  CONST
        LF = ''(10);    { Line feed }

  BEGIN
    WRITELN ( Clear_screen );
    WRITELN ( 'This program calculates a temperature in',
              ' FAHRENHEIT from a' );
    WRITELN ( 'value in CELSIUS input by the user', LF, LF );
    WRITELN ( 'You will be prompted for input', LF, LF );
  END;

```

```
{ ------------------------------------------------------------ }

PROCEDURE Prompt_user;

  BEGIN
        WRITE ( Prompt );
  END;

{ ------------------------------------------------------------ }

FUNCTION Read_Celsius_value : REAL;

  VAR
        Temporary : REAL;

  BEGIN
        READ ( Temporary );   READLN;
        Read_Celsius_value := Temporary;
  END;

{ ------------------------------------------------------------ }

FUNCTION Calculate_Fahrenheit ( C_value : REAL ) : REAL ;

  BEGIN
        Calculate_Fahrenheit := C_value * Conversion_factor +
                                Fahrenheit_zero;
  END;

{ ------------------------------------------------------------ }

PROCEDURE Write_results ( C, F : REAL );

  BEGIN
        WRITELN ( Celsius_input,    C :1:1,
                  Fahrenheit_value, F :1:1 );
  END;

{ ------------------------------------------------------------ }

BEGIN   {  Body of program  }

        Describe_program;
        Prompt_user;
        Celsius    := Read_Celsius_value;
        Fahrenheit := Calculate_Fahrenheit ( Celsius );
        Write_results ( Celsius, Fahrenheit );
END.

END { of module };
```

```
{-----------------------------------------------------------------------
SOURCE:          EXPORT_1.PAS

COMPILE:         $ EPASCAL EXPORT_1

USED BY:         IMPORT_1.PAS
-----------------------------------------------------------------------}

MODULE EXPORT_1 [IDENT ('V1.000')];

{  Outer level declarations  }
{  The exporting is implied in the outer level declarations
   of a module }

EXPORT
        Clear_screen,
        Prompt,
        Celsius_input,
        Fahrenheit_value,
        Conversion_factor,
        Fahrenheit_zero;

CONST
        Clear_screen      = ''(27)'[2J'(27)'[H';
                            { translates to <ESC>[2J<ESC>[H }
        Prompt            = 'Please enter a temperature in Celsius: ';
        Celsius_input     = 'The Celsius temperature input was ';
        Fahrenheit_value  = ' and its Fahrenheit equivalent is ';
        Conversion_factor =  1.8;  { result of 9.0 / 5.0 }
        Fahrenheit_zero   = 32.0;

END.
```

B.4.1 Running IMPORT_1

IMPORT_1 can be built using the command procedure of the same name:

```
$       ! IMPORT_1.COM
$       ! Command procedure to build the VAXELN module
$       !       IMPORT_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile_exporter
$ !
$       SET VERIFY
$         EPASCAL /LIST EXPORT_1
$         EPASCAL /LIST IMPORT_1, EXPORT_1 /MODULE
$         LINK IMPORT_1, EXPORT_1
$       GOTO Switch_off_verify
$ !
$ Compile_exporter:
$       SET VERIFY
$       EPASCAL -
                /LIST -
                EXPORT_1     ! contains exported symbols for import
$ !                                   by IMPORT_1
$ Compile_importer:
$       EPASCAL -
                /LIST -
                /DEBUG IMPORT_1  -
                ,EXPORT_1  /MODULE
$ Link_importer_exporter:
$       LINK -
                /DEBUG IMPORT_1  -
                ,EXPORT_1  -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT IMPORT_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

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

```
{-----------------------------------------------------------------------
SOURCE:         INCLUDE_1.PAS

PURPOSE:        This module converts a temperature in CELSIUS to
                FAHRENHEIT for user input from a terminal.
                It demonstrates the INCLUDE statement in VAXELN
                Pascal

COMPILE:        $ EPASCAL /DEBUG INCLUDE_1, VAXELN-MODULES /LIBRARY

LINK:           $ LINK   /DEBUG INCLUDE_1, VAXELN-MODULES /LIBRARY -
                _$ ,ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT INCLUDE_1

NOTES:          1) Command procedure INCLUDE_1.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
----------------------------------------------------------------------}
MODULE Temperature_converter [IDENT ('V1.000')];

INCLUDE INCLUDE_1_DEFS;    { In the object library: VAXELN-MODULES }

PROGRAM Convert_C_to_F ( INPUT, OUTPUT );

  VAR
        Celsius,
        Fahrenheit      : REAL;

{ ------------------------------------------------------------------ }

PROCEDURE Describe_program;

  CONST
        LF = ''(10);    { Line feed }

  BEGIN
    WRITELN ( Clear_screen );
    WRITELN ( 'This program calculates a temperature in',
              ' FAHRENHEIT from a' );
    WRITELN ( 'value in CELSIUS input by the user', LF, LF );
    WRITELN ( 'You will be prompted for input', LF, LF );
  END;

```

```
{ ------------------------------------------------------------------ }

PROCEDURE Prompt_user;

  BEGIN
        WRITE ( Prompt );
  END;

{ ------------------------------------------------------------------ }

FUNCTION Read_Celsius_value : REAL;

  VAR
        Temporary : REAL;

  BEGIN
        READ ( Temporary );   READLN;
        Read_Celsius_value := Temporary;
  END;

{ ------------------------------------------------------------------ }

FUNCTION Calculate_Fahrenheit ( C_value : REAL ) : REAL ;

  BEGIN
        Calculate_Fahrenheit := C_value * Conversion_factor +
                                Fahrenheit_zero;
  END;

{ ------------------------------------------------------------------ }

PROCEDURE Write_results ( C, F : REAL );

  BEGIN
        WRITELN ( Celsius_input,    C :1:1,
                  Fahrenheit_value, F :1:1 );
  END;

{ ------------------------------------------------------------------ }

BEGIN   {  Body of program  }

        Describe_program;
        Prompt_user;
        Celsius    := Read_Celsius_value;
        Fahrenheit := Calculate_Fahrenheit ( Celsius );
        Write_results ( Celsius, Fahrenheit );
END.

END { of module Temperature converter };
```

The file INCLUDE_1_DEFS.PAS is listed below:

```
{----------------------------------------------------------------------
SOURCE:         INCLUDE_1_DEFS.PAS

COMPILE:        $ EPASCAL /DEBUG INCLUDE_1_DEFS

USED BY:        INCLUDE_1.PAS
----------------------------------------------------------------------}

MODULE INCLUDE_1_DEFS [IDENT ('V1.000')];

{  Outer level declarations  }
{  The exporting is implied in the outer level
   declarations of a module }

EXPORT
        Clear_screen,
        Prompt,
        Celsius_input,
        Fahrenheit_value,
        Conversion_factor,
        Fahrenheit_zero;

CONST
        Clear_screen      = ''(27)'[2J'(27)'[H';
                            { translates to <ESC>[2J<ESC>[H }
        Prompt            = 'Please enter a temperature in Celsius: ';
        Celsius_input     = 'The Celsius temperature input was ';
        Fahrenheit_value  = ' and its Fahrenheit equivalent is ';
        Conversion_factor =  1.8;  { result of 9.0 / 5.0 }
        Fahrenheit_zero   = 32.0;

END.
```

B.5.1 Running INCLUDE_1

INCLUDE_1 can be built using the command procedure of the same name:

```
$       ! INCLUDE_1.COM
$       ! Command procedure to build the VAXELN module
$       ! INCLUDE_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile_Includer
$ !
$       SET VERIFY
$       EPASCAL /LIST INCLUDE_1, VAXELN-MODULES /LIBRARY
$       LINK INCLUDE_1, VAXELN-MODULES /LIBRARY
$       GOTO Switch_off_verify
$ !
$ Compile_Includer:
$       SET VERIFY
$       EPASCAL -
                /LIST -
                /DEBUG INCLUDE_1 -
                ,VAXELN-MODULES/LIBRARY
$ Link_Includer:
$       LINK -
                /DEBUG INCLUDE_1 -
                ,VAXELN-MODULES /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT INCLUDE_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

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
```

B.6 EXAMPLE OF RADIX SPECIFIERS - RADIX_1.PAS

```
{-------------------------------------------------------------------
SOURCE:         RADIX_1.PAS

PURPOSE:        Demonstrates the following facilities:

                        1) Radix specifiers
                        2) BIN function
                        3) HEX function
                        4) OCT function

COMPILE:        $ EPASCAL /LIST /DEBUG RADIX_1

LINK:           $ LINK /DEBUG RADIX_1 -
                _$ ,ELN$:RTLSHARE /LIBRARY -
                _$ ,RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT RADIX_1

NOTES:          1) Command procedure RADIX_1.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
-------------------------------------------------------------------}
MODULE RADIX_1;

CONST
        Binary_value       = %B'01010101 10101010 01010101 10101010';
        Hexadecimal_value  = %X'ABCD DCBA';
        Octal_value        = %O177777;

PROGRAM RADIX_1 ( OUTPUT );

BEGIN
        WRITELN ( 'Binary constant is: ', Binary_value );
        WRITELN ( 'Hex    constant is: ', Hexadecimal_value );
        WRITELN ( 'Octal  constant is: ', Octal_value );

        WRITELN;
        WRITELN ( 'Now with the BIN HEX and OCT functions ... ' );
        WRITELN;
        WRITELN ( 'Binary constant is: ', BIN (Binary_value) );
        WRITELN ( 'Hex    constant is: ', HEX (Hexadecimal_value) );
        WRITELN ( 'Octal  constant is: ', OCT (Octal_value) );

END { of PROGRAM RADIX_1 }.
END { of  MODULE RADIX_1 };
```

B.6.1 Running RADIX_1

RADIX_1 can be built using the command procedure of the same name:

```
$       ! RADIX_1.COM
$       ! Command procedure to build the VAXELN module
$       ! RADIX_1
$ !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST RADIX_1
$         LINK RADIX_1
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  RADIX_1
$ Link:
$               @ELN_LINK_1     RADIX_1
$ System_build:
$               @ELN_EBUILD_1   RADIX_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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
```

B.7 EXAMPLE OF SIZE ATTRIBUTES - ATTRIBS_1.PAS

```
{-----------------------------------------------------------------
SOURCE:          ATTRIBS_1.PAS

PURPOSE:         Demonstrates the following attributes and facilities:

                         1) POS attribute
                         2) BIT attribute
                         3) Extent expression with 1) and 2) above
                         4) WORD attribute
                         5) Typecasting
                         6) BIN function

COMPILE:         $ EPASCAL /LIST /DEBUG ATTRIBS_1

LINK:            $ LINK /DEBUG ATTRIBS_1 -
                 _$ ,ELN$:RTLSHARE /LIBRARY -
                 _$ ,RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT ATTRIBS_1

NOTES:           1) Command procedure ATTRIBS_1.COM compiles, links and
                    builds this module into a VAXELN system or for
                    running under VMS
-----------------------------------------------------------------}
MODULE ATTRIBS_1;

CONST
        Pair    = 2;   { for use as extent expression }
        Trio    = 3;   { for use as extent expression }

PROGRAM ATTRIBS_1 ( OUTPUT );

TYPE
        Word         = [WORD] -32768..32767;

VAR
        Bit_pattern  : [WORD] PACKED RECORD
           L_S_Bit   : [BIT]                0..1; { least sig. bit }
           Bits_02   : [POS (2), BIT (Trio)] 0..7;
           Bits_08   : [POS (8), BIT (Pair)] 0..3;
           M_S_Bit   : [POS(15), BIT]       0..1; { most sig. bit }
        END;

```

```
BEGIN
        WITH Bit_pattern DO
        BEGIN
            L_S_Bit := 1;
            Bits_02 := 5;
            Bits_08 := 3;
            M_S_Bit := 1;

{   After those assignments Bit_pattern looks like this:

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|  Bit position
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    | 1| 0| 0| 0| 0| 0| 1| 1| 0| 0| 0| 1| 0| 1| 0| 1|  After setting
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |--|              |-----|        |--------|  |--|
      ^                  ^                 ^       ^
  M_S_Bit              Bits_08          Bits_02  L_S_Bit

}
            WRITELN (   ' L_S_Bit: ', L_S_Bit:1 );
            WRITELN (   ' Bits_02: ', Bits_02:1 );
            WRITELN (   ' Bits_08: ', Bits_08:1 );
            WRITELN (   ' M_S_Bit: ', M_S_Bit:1 );

            WRITELN (   'Binary pattern for RECORD is: ',
                        BIN (Bit_pattern::BYTE_DATA)      );
            WRITELN (   'Numeric value is: ', Bit_pattern::Word );
        END;

END { of PROGRAM ATTRIBS_1 }.
END { of  MODULE ATTRIBS_1 };
```

B.7.1 Running ATTRIBS_1

ATTRIBS_1 can be built using the command procedure of the same name:

```
$       ! ATTRIBS_1.COM
$       ! Command procedure to build the VAXELN
$       ! module ATTRIBS_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST ATTRIBS_1
$         LINK ATTRIBS_1
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  ATTRIBS_1
$ Link:
$               @ELN_LINK_1     ATTRIBS_1
$ System_build:
$               @ELN_EBUILD_1   ATTRIBS_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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

```
{---------------------------------------------------------------------
SOURCE:          TYPECAST_1.PAS

PURPOSE:         Demonstrates the following facilities:

                         1) Typecasting
                         2) BIN function
                         3) Size attributes

COMPILE:         $ EPASCAL /LIST /DEBUG TYPECAST_1

LINK:            $ LINK /DEBUG TYPECAST_1 -
                 _$ ,ELN$:RTLSHARE /LIBRARY -
                 _$ ,RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT TYPECAST_1

NOTES:           1) Command procedure TYPECAST_1.COM compiles, links and
                    builds this module into a VAXELN system or for
                    running under VMS
---------------------------------------------------------------------}
MODULE  Typecaster [IDENT('V1.000')];

PROGRAM Typecaster ( INPUT, OUTPUT );

TYPE
        Byte_record = [BYTE] PACKED RECORD
                        B1 : [POS(0), BIT(5)] 0..31;
                        UC : [BIT(1)]    0..1;
                        B3 : [BIT(2)]    0..3;
        END;

{
                      Byte_record
               +--+--+--+--+--+--+--+--+
               | 7| 6| 5| 4| 3| 2| 1| 0|  Bit position
               +--+--+--+--+--+--+--+--+
               |-B3--|UC|------B1------|
}

VAR
        Alphabetic      : BOOLEAN;
        Bit_pattern     : Byte_record;
        User_char       : CHAR;
BEGIN

   WRITE  ( 'Please enter character: ' );
   READLN ( User_char );

```

```
{ Check if its in the alphabets }
        Alphabetic := User_char IN [ 'A'..'Z', 'a'..'z' ];

{ Typecast it to the byte record }
        Bit_pattern::BYTE_DATA := User_char::BYTE_DATA;

        WRITELN ( 'Bit pattern before: ', BIN (User_char::BYTE_DATA) );

{ If it IS lowercase alphabetic bit 5 will be set.
  Clear bit 5. That subtracts 32 from the byte value }

        IF Alphabetic THEN
            IF ODD    ( Bit_pattern.UC )
                THEN  BEGIN
                        WRITELN ( 'Lowercase character' );
                        Bit_pattern.UC := 0;
                        User_char::BYTE_DATA := Bit_pattern::BYTE_DATA;
                        WRITELN ( User_char, ' now uppercase' );
                      END

                ELSE  BEGIN
                        WRITELN ( 'Uppercase character' );
                      END;

        WRITELN ( 'Bit pattern after : ', BIN (User_char::BYTE_DATA) );

END { of PROGRAM Typecaster }.
END { of  MODULE Typecaster };
```

B.8.1 Running TYPECAST_1

TYPECAST_1 can be built using the command procedure of the same name:

```
$       ! TYPECAST_1.COM
$       ! Command procedure to build the VAXELN module TYPECAST_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST TYPECAST_1
$         LINK TYPECAST_1
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  TYPECAST_1
$ Link:
$               @ELN_LINK_1     TYPECAST_1
$ System_build:
$               @ELN_EBUILD_1   TYPECAST_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
program TYPECAST_1 /debug
```

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

```
{-----------------------------------------------------------------------
SOURCE:          SCOPE_1.PAS

PURPOSE:         Demonstrates a simple MODULE and PROGRAM in VAXELN
                 Pascal with outer level declaration of a CONST

COMPILE:         $ EPASCAL /LIST /DEBUG SCOPE_1

LINK:            $ LINK /DEBUG SCOPE_1, ELN$:RTLSHARE /LIBRARY, -
                 _$ RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT SCOPE_1

NOTES:           1) Command procedure SCOPE_1.COM compiles, links and
                    builds this module into a VAXELN system or for
                    running under VMS
-----------------------------------------------------------------------}
MODULE SCOPE_1 [IDENT ('V1.000')];

    CONST        { This outer level declaration is EXPORTed explicitly }
        Greeting = 'A simple program to test VAXELN';

PROGRAM SCOPE_1 (OUTPUT);

    BEGIN
        WRITELN (  Greeting );

END { of PROGRAM SCOPE_1 }.

END { of MODULE  SCOPE_1 };
```

B.9.1 Running SCOPE_1

SCOPE_1 can be built using the command procedure of the same name:

```
$       ! SCOPE_1.COM
$       ! Command procedure to build the VAXELN module SCOPE_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST SCOPE_1
$         LINK SCOPE_1
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  SCOPE_1
$ Link:
$               @ELN_LINK_1     SCOPE_1
$ System_build:
$               @ELN_EBUILD_1   SCOPE_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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

```
{-----------------------------------------------------------------
SOURCE:         SCOPE_2.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal with outer level declaration of a CONST

COMPILE:        $ EPASCAL /LIST /DEBUG SCOPE_2

LINK:           $ LINK /DEBUG SCOPE_2, ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SCOPE_2

NOTES:          1) Command procedure SCOPE_2.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
-----------------------------------------------------------------}
MODULE SCOPE_2 [IDENT ('V1.000')];

  CONST   { This outer level declaration is EXPORTed explicitly }
        Greeting = 'A simple program to test VAXELN';

PROGRAM SCOPE_2 (OUTPUT);

  CONST
        Greeting = 'This is an inner declaration of the same name';

    BEGIN

        WRITELN (  Greeting );
        { prints "This is an inner declaration of the same name" }

END { of PROGRAM SCOPE_2 }.
END { of  MODULE SCOPE_2 };
```

B.10.1 Running SCOPE_2

SCOPE_2 can be built using the command procedure of the same name:

```
$       ! SCOPE_2.COM
$       ! Command procedure to build the VAXELN module SCOPE_2
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST SCOPE_2
$         LINK SCOPE_2
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  SCOPE_2
$ Link:
$               @ELN_LINK_1     SCOPE_2
$ System_build:
$               @ELN_EBUILD_1   SCOPE_2
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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

```
{------------------------------------------------------------------
SOURCE:         SCOPE_3.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal with outer level declaration of a CONST and
                PROCEDURE

COMPILE:        $ EPASCAL /LIST /DEBUG SCOPE_3

LINK:           $ LINK /DEBUG SCOPE_3, ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SCOPE_3

NOTES:          1) Command procedure SCOPE_3.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
------------------------------------------------------------------}
MODULE SCOPE_3 [IDENT ('V1.000')];

   CONST        { This outer level declaration is EXPORTed explicitly }
        Greeting = 'A simple program to test VAXELN';

{------------------------------------------------------------------}
PROCEDURE Show_scope;

    BEGIN
        WRITELN ( 'This is from procedure SHOW_SCOPE about ',
                  'to write GREETING' );
        WRITELN ( Greeting );
    END { PROCEDURE Show_scope };
{------------------------------------------------------------------}

PROGRAM SCOPE_3 (OUTPUT);

  CONST
        Greeting = 'This is an inner declaration of the same name';

    BEGIN

        WRITELN (  Greeting );
        { prints "This is an inner declaration of the same name" }
        Show_scope;

END { of PROGRAM SCOPE_3 }.
END { of  MODULE SCOPE_3 };
```

B.11.1 Running SCOPE_3

SCOPE_3 can be built using the command procedure of the same name:

```
$       ! SCOPE_3.COM
$       ! Command procedure to build the VAXELN module SCOPE_3
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST SCOPE_3
$         LINK SCOPE_3
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  SCOPE_3
$ Link:
$               @ELN_LINK_1     SCOPE_3
$ System_build:
$               @ELN_EBUILD_1   SCOPE_3
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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

```
{-------------------------------------------------------------------
SOURCE:         SCOPE_3A.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal with outer level declaration of a CONST and a
                PROCEDURE declared in the program.

COMPILE:        $ EPASCAL /LIST /DEBUG SCOPE_3A

LINK:           $ LINK /DEBUG SCOPE_3A, ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SCOPE_3A

NOTES:          1) Command procedure SCOPE_3A.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
------------------------------------------------------------------}
MODULE SCOPE_3A [IDENT ('V1.000')];

   CONST        { This outer level declaration is EXPORTed explicitly }
        Greeting = 'A simple program to test VAXELN';
{-----------------------------------------------------------------}

PROGRAM SCOPE_3A (OUTPUT);

  CONST
        Greeting = 'This is an inner declaration of the same name';

{-----------------------------------------------------------------}
PROCEDURE Show_scope;

    BEGIN
        WRITELN ( 'This is from procedure SHOW_SCOPE about ',
                  'to write GREETING' );
        WRITELN ( Greeting );
    END { PROCEDURE Show_scope };

{-----------------------------------------------------------------}
    BEGIN

        WRITELN (  Greeting );
        { prints "This is an inner declaration of the same name" }
        Show_scope;

END { of PROGRAM SCOPE_3A }.
END { of  MODULE SCOPE_3A };
```

B.12.1 Running SCOPE_3A

SCOPE_3A can be built using the command procedure of the same name:

```
$        ! SCOPE_3A.COM
$        ! Command procedure to build the VAXELN module SCOPE_3A
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        INQUIRE -
                 /NOPUNCTUATION  Running_under_VMS -
                 "Do you wish to run this program under VMS [Y/N]: "
$        IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$        SET VERIFY
$          EPASCAL /LIST SCOPE_3A
$          LINK SCOPE_3A
$        GOTO Switch_off_verify
$ !
$ Compile:
$                @ELN_COMPILE_1  SCOPE_3A
$ Link:
$                @ELN_LINK_1     SCOPE_3A
$ System_build:
$                @ELN_EBUILD_1   SCOPE_3A
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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 availablity of full procedures.

```
{-----------------------------------------------------------------------
SOURCE:         SEPARATE_PROG_1.PAS

PURPOSE:        Demonstrates a simple MODULE and PROGRAM in VAXELN
                Pascal with outer level declaration of a PROCEDURE
                The procedure body is in a source file compiled
                separately

NEEDS:          1) SEPARATE_MOD_1.OBJ

COMPILE:        $ EPASCAL /LIST /DEBUG SEPARATE_MOD_1
                $ EPASCAL /LIST /DEBUG SEPARATE_BODY_1, -
                _$ SEPARATE_MOD_1/MOD
                $ EPASCAL /LIST /DEBUG SEPARATE_PROG_1, -
                _$ SEPARATE_MOD_1/MODULE

LINK:           $ LINK /DEBUG SEPARATE_PROG_1, SEPARATE_BODY_1, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SEPARATE_PROG_1

NOTES:          1) Command procedure SEPARATE_PROG_1.COM compiles,
                   links and builds this module into a system.
                   It may also be built to run under VMS using a
                   simple LINK command
-----------------------------------------------------------------------}
MODULE SEPARATE_PROG_1 [IDENT ('V1.000')];

INCLUDE
        SEPARATE_MOD_1; { PROCEDURE Show_scope; SEPARATE; }

PROGRAM SEPARATE_PROG_1 (OUTPUT);

  CONST
        Greeting = 'This is an inner declaration of the same name';

    BEGIN

        WRITELN (  Greeting );
        { prints "This is an inner declaration of the same name" }
        Show_scope;

END { of PROGRAM SEPARATE_PROG_1 }.
END { of  MODULE SEPARATE_PROG_1 };
```

```
MODULE SEPARATE_MOD_1 [IDENT ('V1.000')];

PROCEDURE Show_scope; SEPARATE;

END; { MODULE SEPARATE_MOD_1 }
```

```
MODULE SEPARATE_BODY_1 [IDENT ('V1.000')];

{SOURCE:   SEPARATE_BODY_1.PAS
 COMPILE:  $ EPASCAL SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
}
PROCEDURE_BODY Show_scope;

CONST
        Greeting = 'A simple program to test VAXELN';

    BEGIN
        WRITELN ( 'This is from procedure SHOW_SCOPE about ',
                  'to write GREETING' );
        WRITELN ( Greeting );
    END { PROCEDURE Show_scope };

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:

```
$       ! SEPARATE_PROG_1.COM
$       ! Command procedure to build the VAXELN
$       ! module SEPARATE_PROG_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile_defs
$ !
$       SET VERIFY
$         EPASCAL /LIST SEPARATE_MOD_1
$         EPASCAL /LIST SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
$         EPASCAL /LIST SEPARATE_PROG_1, SEPARATE_MOD_1 /MODULE
$         LINK SEPARATE_PROG_1, SEPARATE_BODY_1
$       GOTO Switch_off_verify
$ !
$ Compile_defs:
$       EPASCAL -
                /LIST -
                /DEBUG SEPARATE_MOD_1
$       EPASCAL -
                /LIST -
                /DEBUG SEPARATE_BODY_1, SEPARATE_MOD_1 /MODULE
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG SEPARATE_PROG_1, SEPARATE_MOD_1 /MODULE
$ Link:
$       LINK -
                /DEBUG SEPARATE_PROG_1, SEPARATE_BODY_1 -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT SEPARATE_PROG_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
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

```
{----------------------------------------------------------------
SOURCE:         AGGREG_1.PAS

PURPOSE:        Demonstrates aggregate initialization on an array

COMPILE:        $ EPASCAL /LIST /DEBUG AGGREG_1

LINK:           $ LINK /DEBUG AGGREG_1 -
                _$ ,ELN$:RTLSHARE /LIBRARY -
                _$ ,RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT AGGREG_1

NOTES:          1) Command procedure AGGREG_1.COM compiles, links and
                   builds this module into a VAXELN system or for
                   running under VMS
----------------------------------------------------------------}
MODULE Aggregate [IDENT ('V1.000')];

TYPE
        Chess_pieces = (QR,QN,QB,Q,K,KB,KN,KR,P,E);

{ here is the aggregate initializer for the array - the technique
  can be used on RECORDs as well.

  The author has 'lifted' this one from the VAX Pascal Language
  Reference Manual to whose authors he is grateful
}

VAR
        Board    : ARRAY[1..8,QR..KR] OF Chess_pieces :=
                        ((QR,QN,QB,Q,K,KB,KN,KR),
                        (8 OF P),
                        4 OF (8 OF E),
                        (8 OF P),
                        (QR,QN,QB,Q,K,KB,KN,KR));

```

```
PROGRAM Chessboard (OUTPUT);

VAR
        Rows   : 1..8;
        Pieces : QR..KR;

BEGIN
        FOR Rows := 1 TO 8 DO
          BEGIN
           WRITELN;
           WRITE(' ':25);
           FOR Pieces := QR TO KR DO WRITE ( Board[Rows,Pieces] );
          END;
        WRITELN;

END {of PROGRAM}.
END {of MODULE};
```

B.14.1 Running AGGREG_1

This module can be built using AGGREG_1.COM:

```
$       ! AGGREG_1.COM
$       ! Command procedure to build the VAXELN
$       ! module AGGREG_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$       SET VERIFY
$         EPASCAL /LIST AGGREG_1
$         LINK AGGREG_1
$       GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  AGGREG_1
$ Link:
$               @ELN_LINK_1     AGGREG_1
$ System_build:
$               @ELN_EBUILD_1   AGGREG_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EDEBUG 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

```
MODULE Check_status_and_report [IDENT ('V1.100')];

{
  This module checks status returns and generates VMS-like error
  messages when non-ODD status codes are returned

  Modification history:

  1) 09 February 1987: $OTSMSG line added to INCLUDE heading

}

```

```
INCLUDE
            $GET_MESSAGE_TEXT,
            $ELNMSG,
            $PASCALMSG,
            $KERNELMSG,
            $OTSMSG;            { Added 9-Feb-1987 }
VAR
        Text_of_error           : VARYING_STRING (255);

PROCEDURE Check_status_and_report ( Returned_status : INTEGER;
                                   Call_name : VARYING_STRING (80) );
    BEGIN
        IF NOT ODD ( Returned_status ) THEN
          BEGIN
            WRITELN ( 'Unsuccessful call to: ',
                      Call_name,
                      ' status was: ',
                      Returned_status : 1 );

            ELN$GET_STATUS_TEXT ( Returned_status,
                                  [ STATUS$FACILITY,
                                    STATUS$SEVERITY,
                                    STATUS$IDENT,
                                    STATUS$TEXT ],
                                  Text_of_error );
            WRITELN ( Text_of_error );
          END;
    END;
END;
```

Now the listing of the module:

```
{_____________________________________________________________________
SOURCE:         MOD-PROG-PROC-1.PAS

PURPOSE:        Demonstrates a simple MODULE, PROGRAM and PROCESS
                in VAXELN

COMPILE:        $ EPASCAL /LIST /DEBUG MOD-PROG-PROC-1, -
                _$ VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG MOD-PROG-PROC-1, -
                _$ VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT MOD-PROG-PROC-1

NOTES:          1) Command procedure MOD-PROG-PROC-1.COM compiles,
                   links and builds this module into a system
______________________________________________________________________
}

MODULE Create_process_01 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

{ These outer-level declarations of CONST and PROCESS_BLOCK are
  EXPORTed implicitly
}
    CONST
        Master_process_greeting = 'This is from the master process';
        Process_greeting        = 'This is from the created process';

{-------------- PROCESS BLOCK ---------------}

PROCESS_BLOCK Simple_process ( Value_passed : INTEGER );

   BEGIN
    WRITELN ( Process_greeting );
    WRITELN ( 'Value passed to process is: ', Value_passed :1 );

   END { PROCESS BLOCK };
```

```
{-------------------- PROGRAM BLOCK --------------------}

     PROGRAM Process_create (OUTPUT);

        VAR
            Process_identity    : PROCESS;

            Returned_status,
            Number              : INTEGER;

{-------------------------------------------------------}

PROCEDURE Issue_master_process_greeting;

    BEGIN
        WRITELN ( Master_process_greeting );
    END;

{-------------------------------------------------------}

PROCEDURE Create_process_and_wait;
  BEGIN
    CREATE_PROCESS ( Process_identity,
                     Simple_process,
                     Number,
                     STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_PROCESS' );

    WAIT_ANY ( Process_identity,
               STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

END { PROCEDURE DECLARATIONS };

{ MAIN program start }

  BEGIN
    Issue_master_process_greeting;
       Number := 1986;

    Create_process_and_wait;

END {PROGRAM BLOCK}.
END {MODULE};
```

C.1.1 Running MOD-PROG-PROC-1.PAS

MOD-PROG-PROC-1.PAS can be built using the command procedure of the same name:

```
$       ! MOD-PROG-PROC-1.COM
$       ! Command procedure to build the VAXELN
$       ! module MOD-PROG-PROC-1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_01:
$       EPASCAL -
                /LIST -
                /DEBUG MOD-PROG-PROC-1, VAXELN-MODULES /LIBRARY
$ Link_01:
$       LINK -
                /DEBUG MOD-PROG-PROC-1 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT MOD-PROG-PROC-1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD is listed below:

```
characteristic /noconsole
program MOD-PROG-PROC-1 /debug
```

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

```
{____________________________________________________________________
SOURCE:         MOD-PROG-PROC-1A.PAS

PURPOSE:        Demonstrates a simple MODULE, PROGRAM and PROCESS
                in VAXELN

NEEDS:          VAXELN-MODULES.OLB

COMPILE:        $ EPASCAL /LIST /DEBUG MOD-PROG-PROC-1A, -
                _$ VAXELN-MODULES /LIB

LINK:           $ LINK /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIB, -
                _$ ELN$:RTLSHARE /LIBRARY, RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$ (OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT MOD-PROG-PROC-1A

NOTES:          1) Command procedure MOD-PROG-PROC-1A.COM compiles,
                   links and builds this module into a system
_____________________________________________________________________}
MODULE Create_process_01 [IDENT ('V1.000')];

INCLUDE MOD_PROG_PROC_1A_DEFS;  { In library VAXELN-MODULES }

PROGRAM Process_create (OUTPUT);

        { MAIN program start }

                BEGIN
                        Issue_master_process_greeting;
                        Number := 1986;
                        Create_process_and_wait;

                END {PROGRAM}.
END {MODULE};
```

The definition module looks like this:

```
MODULE MOD_PROG_PROC_1A_DEFS [IDENT ('V1.000')];
{
  SOURCE:    MOD-PROG-PROC-1A-DEFS.PAS
  COMPILE:   $ EPASCAL MOD-PROG-PROC-1A-DEFS.PAS, -
             _$ VAXELN-MODULES /LIBRARY

  These outer-level declarations of CONST and PROCESS_BLOCK are
  EXPORTed implicitly
}

INCLUDE
        Check_status_and_report;

CONST
        Master_process_greeting = 'This is from the master process';
        Process_greeting        = 'This is from the created process';

VAR
        Process_identity        : PROCESS;
        Returned_status,
        Number                  : INTEGER;

{--------------- PROCESS BLOCK ---------------}

PROCESS_BLOCK Simple_process ( Value_passed : INTEGER );

   BEGIN
    WRITELN ( Process_greeting );
    WRITELN ( 'Value passed to process is: ', Value_passed :1 );

END {PROCESS BLOCK};
{---------------------------------------------}

PROCEDURE Issue_master_process_greeting;

    BEGIN
        WRITELN ( Master_process_greeting );
    END;

```

```
{---------------------------------------------}

PROCEDURE Create_process_and_wait;
  BEGIN
    CREATE_PROCESS ( Process_identity,
                     Simple_process,
                     Number,
                     STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_PROCESS' );

    WAIT_ANY ( Process_identity,
               STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

END {PROCEDURE DECLARATIONS};
END {MODULE}.
```

C.2.1 Running MOD-PROG-PROC-1A

This module can be built using the command procedure of the same name:

```
$        ! MOD-PROG-PROC-1A.COM
$        ! Command procedure to build the VAXELN
$        ! module MOD-PROG-PROC-1A
$ !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ !
$ Compile:
$        EPASCAL -
                 /LIST -
                 /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIB
$ Link:
$        LINK -
                 /DEBUG MOD-PROG-PROC-1A, VAXELN-MODULES /LIBRARY -
                 ,ELN$:RTLSHARE /LIBRARY -
                 ,ELN$:RTL /LIBRARY /INCLUDE= -
                 (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$        EBUILD -
                         /NOEDIT MOD-PROG-PROC-1A
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file looks like this:

```
        characteristic /noconsole
        program MOD-PROG-PROC-1A /debug
```

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:

```
{-----------------------------------------------------------------
SOURCE:         ERRORS_1.PAS

PURPOSE:        To demonstrate error message handling using a user-
                defined routine to print messages

COMPILE:        $ EPASCAL /LIST /DEBUG ERRORS_1, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG ERRORS_1, VAXELN-MODULES /LIBRARY,
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT ERRORS_1

NOTES:          1) Command procedure ERRORS_1.COM compiles, links and
                   builds this module into a VAXELN system
-----------------------------------------------------------------}
MODULE Errors_01 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report; { user-defined message writer }

{------------------- PROGRAM BLOCK -------------------}

PROGRAM Errors (OUTPUT);

    VAR
        Flag                : EVENT;
        Returned_status     : INTEGER;

{ MAIN program start }

    BEGIN
        CREATE_EVENT    (       Flag,
                                EVENT$CLEARED,
                                STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_EVENT' );

```

```
        DELETE          (       Flag,
                                STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                    'DELETE' );

{ Force error condition by waiting for non-existent object }

        WAIT_ANY        (       Flag,
                                STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                    'WAIT_ANY' );

END { of PROGRAM }.
END { of  MODULE };
```

C.3.1 Running ERRORS_1

This module can be built using the command procedure of the same name:

```
$       ! ERRORS_1.COM
$       ! Command procedure to compile and link the VAXELN
$       ! module ERRORS_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG ERRORS_1 -
                ,VAXELN-MODULES /LIBRARY
$ Link:
$       LINK -
                /DEBUG ERRORS_1 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT ERRORS_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

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

```
{-----------------------------------------------------------------
SOURCE:         ERRORS_2.PAS

PURPOSE:        To demonstrate error message handling

NEEDS:          ERRORS_2_MSG.MSG

COMPILE:        $ EPASCAL /LIST /DEBUG ERRORS_2,
                _$  ELN$:RTLOBJECT /LIBRARY

                $ MESSAGE /OBJECT ERRORS_2_MSG

LINK:           $ LINK /DEBUG ERRORS_2, ERRORS_2_MSG, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT ERRORS_2

NOTES:          1) Command procedure ERRORS_2.COM compiles, links and
                   builds this module into a VAXELN system
-----------------------------------------------------------------}
MODULE Errors_02 [IDENT ('V1.000')];

INCLUDE
        $GET_MESSAGE_TEXT;

VAR
 { the symbol "User_obj_deleted" is defined in ERRORS_2_MSG.MSG }

        KER$_BAD_VALUE,
        User_Obj_deleted : [EXTERNAL, VALUE] INTEGER;

{ this SET type is defined in $GET_MESSAGE_TEXT }

        Message_fields  : SET OF Get_status_fields := [];


```

```
{------------------- PROGRAM BLOCK -------------------}

PROGRAM Errors_02 (OUTPUT);

    VAR
        Flag                    : EVENT;
        Returned_status         : INTEGER;
        Mess_text               : VARYING_STRING(255);

{ MAIN program start }

    BEGIN
        CREATE_EVENT    (       Flag,
                                EVENT$CLEARED,
                                STATUS := Returned_status );

        IF NOT ODD ( Returned_status )
                THEN WRITELN ( 'CREATE_EVENT problem, status was: ',
                                Returned_status:1 );

        DELETE          (       Flag,
                                STATUS := Returned_status );

        IF NOT ODD ( Returned_status )
                THEN WRITELN ( 'DELETE problem, status was: ',
                                Returned_status:1 );

{ Force error condition by waiting for non-existent object }

        WAIT_ANY        (       Flag,
                                STATUS := Returned_status );

        IF Returned_status = KER$_BAD_VALUE
          THEN
            BEGIN
                ELN$GET_STATUS_TEXT ( User_obj_deleted,
                                      Message_fields,
                                      Mess_text );
                WRITELN ( Mess_text );
            END;

END { of PROGRAM }.
END { of  MODULE };
```

C.4.1 Running ERRORS_2

This module can be built using the command procedure of the same name:

```
$       ! ERRORS_2.COM
$       ! Command procedure to compile and link the VAXELN
$       ! module ERRORS_2
$       !
$       !
$ !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG ERRORS_2 -
                , ELN$:RTLOBJECT /LIBRARY
$ Message:
        MESSAGE -
                /OBJECT ERRORS_2_MSG
$ Link:
$       LINK -
                /DEBUG ERRORS_2, ERRORS_2_MSG -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT ERRORS_2
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The file ERRORS_2_MSG.MSG looks like this:

```
                .TITLE  VAXELN_demo

                .FACILITY               USER,  1


          .SEVERITY             SEVERE
          OBJ_DELETED < Wait failed because the object did not exist >

          .END
```

The .DAT file looks like this:

```
characteristic /noconsole /nonetwork /nofile /noserver
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.

```
{
SOURCE:         TIME_1.PAS

PURPOSE:        Demonstrates the time routines in VAXELN

COMPILE:        $ EPASCAL /LIST /DEBUG TIME_1

LINK:           $ LINK /DEBUG TIME_1, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT TIME_1

NOTES:          1) Command procedure TIME_1.COM compiles, links and
                   builds this module into a system

}
```

```
MODULE All_times [IDENT ('V1.000')];

CONST
        Interval                        = '0007 12:30:30.00';
VAR
        From_users_input_time           : VARYING_STRING (23);

        At_this_moment,
        Binary_interval,
        New_time,
        The_current_binary_time_is      : LARGE_INTEGER;

        Status_from_call_to_GET_TIME,
        Status_from_call_to_GET_TIME_1,
        Status_from_call_to_SET_TIME    : INTEGER;

        Time_elements                   : TIME_RECORD;

{ ------------------------------------------------------------- }

PROCEDURE Prompt_user_for_time;

  BEGIN
        WRITELN ( 'Please enter time e.g. ',
                  '12-Jan-1987 15:47:30.00 > ' );
        READLN  ( From_users_input_time );
  END;

{ ------------------------------------------------------------- }

PROCEDURE And_get_the_binary_time;

 BEGIN
   The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
 END;

{ ------------------------------------------------------------- }

PROCEDURE Set_the_system_time;

 BEGIN
   SET_TIME     ( The_current_binary_time_is,
                  STATUS := Status_from_call_to_SET_TIME );

   IF
        NOT ODD ( Status_from_call_to_SET_TIME )
   THEN
        WRITELN (  'Call to SET_TIME failed with status: ',
                   Status_from_call_to_SET_TIME
                );
 END;

```

```
{ ------------------------------------------------------------ }

PROCEDURE And_read_time_fields;

 BEGIN
   GET_TIME      ( At_this_moment,
                   STATUS := Status_from_call_to_GET_TIME );

   IF
        NOT ODD ( Status_from_call_to_GET_TIME )
   THEN
        WRITELN (  'Call to GET_TIME failed with status: ',
                   Status_from_call_to_GET_TIME
                );

        Time_elements := TIME_FIELDS    ( At_this_moment );

        WRITELN ( 'The current TIME_FIELDS are: ' );

        WITH Time_elements DO
          WRITELN ( Hour       :2, ':',
                    Minute     :2, ':',
                    Second     :2, '.',
                    Hundredth :2, ' on ',
                    Day        :2, '/',
                    Month      :2, '/',
                    Year       :4
                  );
 END;

{ ------------------------------------------------------------ }

PROCEDURE Get_the_binary_for_interval;

 BEGIN
   Binary_interval := TIME_VALUE ( Interval );
 END;

{ ------------------------------------------------------------ }

PROCEDURE Add_interval_to_current_time;

 BEGIN
        { NOTE: delta times are held as negative integers. To
                advance by an interval SUBTRACT that interval! }

```

```
    GET_TIME     ( At_this_moment,
                   STATUS := Status_from_call_to_GET_TIME_1 );

    IF
         NOT ODD ( Status_from_call_to_GET_TIME_1 )
    THEN
         WRITELN (  'Call to GET_TIME failed with status: ',
                     Status_from_call_to_GET_TIME_1
                 );

   New_time :=  At_this_moment - Binary_interval;

  END;

 { ------------------------------------------------------------ }

 PROCEDURE And_find_new_absolute_time;

  BEGIN
         WRITELN  ( 'The new time formed by adding an interval of: ',
                    Interval, ' is: ' );
         WRITELN  ( TIME_STRING ( New_time ) );
  END;

 { ------------------------------------------------------------ }

 PROGRAM Show_times ( INPUT, OUTPUT );

 BEGIN
         Prompt_user_for_time;
           And_get_the_binary_time;

         Set_the_system_time;
           And_read_time_fields;

         Get_the_binary_for_interval;
           Add_interval_to_current_time;

         And_find_new_absolute_time;

 END { of PROGRAM }.
 END { of MODULE  };
```

D.1.1 Running TIME_1

Use the command procedure TIME_1.COM to build this program:

```
$       ! TIME_1.COM
$       ! Command procedure to build the VAXELN module TIME_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_Time:
$       EPASCAL -
                /LIST -
                /DEBUG TIME_1
$ Link_Time:
$       LINK -
                /DEBUG TIME_1 -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT TIME_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program TIME_1 /debug
```

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.

```
{
SOURCE:         TIME_2.PAS

PURPOSE:        Sets the system time then creates a job to run a
                program written in VAX FORTRAN that returns:
                  a) day
                  b) month
                  c) year
                  d) week
                  e) year day
                  f) century day
                for that date

COMPILE:        $ EPASCAL /LIST /DEBUG TIME_2

                $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY

LINK:           $ LINK /DEBUG TIME_2, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

                $ LINK /DEBUG /NOSYSLIB TODAY, -
                _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT TIME_2

NOTES:          1) Command procedure TIME_2.COM compiles, links and
                   builds these programs into a system
________________________________________________________________________
}
MODULE Set_system_time [IDENT ('V1.000')];

VAR
        From_users_input_time           : VARYING_STRING (23);
        Status_from_call_to_SET_TIME    : INTEGER;
        The_current_binary_time_is      : LARGE_INTEGER;

{ ---------------------------------------------------------------- }

PROCEDURE Prompt_user_for_time;

  BEGIN
        WRITELN ( 'Please enter time e.g. ',
                  '12-Dec-1986 15:47:30.00 > ' );
        READLN  ( From_users_input_time );
  END;

```

```
{ ------------------------------------------------------------- }

PROCEDURE And_get_the_binary_time;

 BEGIN
   The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
 END;

{ ------------------------------------------------------------- }

PROCEDURE Set_the_system_time;

 BEGIN
   SET_TIME      ( The_current_binary_time_is,
                   STATUS := Status_from_call_to_SET_TIME );

   IF NOT ODD    ( Status_from_call_to_SET_TIME ) THEN
        WRITELN (  'Call to SET_TIME failed with status: ',
                    Status_from_call_to_SET_TIME );
 END;

{ ------------------------------------------------------------- }

PROGRAM Set_system_time ( INPUT, OUTPUT );

VAR
        Status_returned : INTEGER;
        Jobs_port       : PORT;

BEGIN
        Prompt_user_for_time;
        And_get_the_binary_time;
        Set_the_system_time;

        CREATE_JOB  ( Jobs_port,
                      'TODAY',
                      STATUS := Status_returned );
           IF NOT ODD ( Status_returned ) THEN
              WRITELN ( 'Call to CREATE_JOB failed with status: ',
                        Status_returned );

END { of PROGRAM }.
END { of MODULE  };
```

The source code for the FORTRAN program TODAY is listed below.

```
*
*    MODULE:         TODAY.FOR
*
*    PURPOSE:        To calculate:
*                      a) Day of week
*                      b) Day of year
*                      c) Day of century
*                      d) Week number (crude version)
*                    for any date after today.
*                    This version is fixed to pick up today's date
*                    but is capable of taking any date from 1770
*
*    COMPILE:        $ FORTRAN /TODAY
*    LINK:           $ LINK /TODAY
*
*    RESTRICTIONS:   Works until 31-Dec-2086
*

      IMPLICIT NONE

      INTEGER        Month_length(12) /31, 28, 31, 30, 31, 30,
     1                                   31, 31, 30, 31, 30, 31/

      INTEGER        Verbal_day_value(7) /1, 2, 3, 21, 22, 23, 31/

      CHARACTER *9   Month_names(12)  /'January',    'February',
     1                                 'March',      'April',
     1                                 'May',        'June',
     1                                 'July',       'August',
     1                                 'September',  'October',
     1                                 'November',   'December'/

      CHARACTER *9   Day_of_week(0:6) /'Sunday',     'Monday',
     1                                 'Tuesday',    'Wednesday',
     1                                 'Thursday',   'Friday',
     1                                 'Saturday'/

      CHARACTER *2   Date_suffix(4)   /'st', 'nd', 'rd', 'th'/

      CHARACTER *8   Time_now
```

```

      INTEGER         Y,
      1               K,
      1               Day,
      1               Day_suffix_no,
      1               Day_of_year,
      1               Week,
      1               Month,
      1               Year,
      1               Century,
      1               Start_of_century,
      1               Century_day,
      1               Grand_day_count,
      1               Weekday

      LOGICAL         Leap_year

*==>> Statement function to determine leap year/no leap year

      Leap_year(Y) =  MOD (Y,   4) .EQ. 0 .AND.
      1               MOD (Y, 100) .NE. 0 .OR.
      1               MOD (Y, 400) .EQ. 0

*==>> Get numeric values of date;
*==>> Program will work correctly until 31 December 2086

      CALL IDATE ( Month, Day, Year )

      IF (Year .LT. 87) THEN
        Year = Year + 2000
      ELSE
        Year = Year + 1900
      END IF

*==>> Initialize year day value;
*==>> Find total days so far this year

      Day_of_year = Day

      DO K = 1, Month-1, 1
      Day_of_year = Day_of_year + Month_length(K)
      END DO

      IF (Month .GT. 2 .AND. Leap_year(Year)) Day_of_year = Day_of_year + 1

*==>> Find week for current year. NOTE: This is not the
*==>> internationally agreed definition of week

      Week = (Day_of_year - 1) / 7 + 1

```

```
*==>> Get century number BUT check not e.g. 1900 - last year of
*==>> 19th century

      Century_day = Day_of_year

      Century = Year / 100
      IF (MOD (Year, 100) .EQ. 0) Century = Century - 1
      Start_of_century = Century * 100 + 1

      DO K = Start_of_century, Year-1, 1
      IF (Leap_year(K)) THEN
        Century_day = Century_day + 366
      ELSE
        Century_day = Century_day + 365
      END IF
      END DO

*==>> Initialize total day count;
*==>> Total days from start date to last year
*==>> N.B 1 Jan 1770 was a MONDAY

      Grand_day_count = Day_of_year

      DO K = 1770, Year-1, 1
      IF (Leap_year(K)) THEN
        Grand_day_count = Grand_day_count + 366
      ELSE
        Grand_day_count = Grand_day_count + 365
      END IF
      END DO

*==>> Find day of week from total day count;
*==>> Determine which of "st", "nd", "rd" or "th" suffixes is
*==>> required

      Weekday = MOD (Grand_day_count, 7)

      Day_suffix_no = 4

      DO K = 1, 7, 1
      IF (Day .EQ. Verbal_day_value(K)) Day_suffix_no = MOD (Day, 10)
      END DO

      CALL Time ( Time_now )

```

```
*==>> Print special graphics box

      WRITE (6, '(//T10, A, A, A)') 27, '(0',
     1'lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk'

      WRITE (6, '(T10, A)')
     1'x                                                              x'

      WRITE (6, '(T10, A, A, A)')
     1'mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj',
     1       27, '(B'

*==>> Print results (up two rows to centre in box)

      WRITE (6, '(T12, A, A, A, A, A, '' '', A, I3, A, '' '',
     1       A, I5, A, A, ''  W'', I3,'', Y'', I4, '', C'', I6, // )')
     1       27, '[2A', 27, '[7m', Time_now,
     1       Day_of_week(Weekday), Day, Date_suffix(Day_suffix_no),
     1       Month_names(Month), Year,
     1       27, '[0m', Week, Day_of_year, Century_day

      END
```

D.2.1 Running TIME_2

Use the command procedure TIME_2.COM to build this system:

```
$       ! TIME_2.COM
$       ! Command procedure to build the EPASCAL module TIME_2
$       ! and the VAX FORTRAN program TODAY and build them into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_Time_2:
$       EPASCAL -
                /LIST -
                /DEBUG TIME_2
$ Compile_Today:
$       FORTRAN -
                /LIST -
                /NOOPTIMIZE -
                /DEBUG TODAY
$ Link_Time_2:
$       LINK -
                /DEBUG TIME_2 -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ Link_Today:
$       LINK -
                /DEBUG -
                /NOSYSLIB TODAY -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT TIME_2
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program TIME_2 /debug
program TODAY /norun /debug
```

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

```
{
SOURCE:         TIME_2A.PAS

PURPOSE:        This program is a variation on TIME_2. This version
                calls INITIALIZATION_DONE instead of creating a job
                to cause TODAY to run.

                The program TODAY written in VAX FORTRAN returns:
                  a) day
                  b) month
                  c) year
                  d) week
                  e) year day
                  f) century day
                for the date

COMPILE:        $ EPASCAL /LIST /DEBUG TIME_2A

                $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY

LINK:           $ LINK /DEBUG TIME_2A, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

                $ LINK /DEBUG /NOSYSLIB TODAY, -
                _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT TIME_2A

NOTES:          1) Command procedure TIME_2A.COM compiles, links and
                   builds these programs into a system

}
MODULE Set_system_time [IDENT ('V1.000')];

VAR
        From_users_input_time          : VARYING_STRING (23);
        Status_from_call_to_SET_TIME   : INTEGER;
        The_current_binary_time_is     : LARGE_INTEGER;

{ ----------------------------------------------------------- }
```

```

PROCEDURE Prompt_user_for_time;

  BEGIN
        WRITELN ( 'Please enter time e.g. ',
                  '12-DEC-1986 15:47:30.00 > ' );
        READLN  ( From_users_input_time );
  END;

{ ------------------------------------------------------------- }

PROCEDURE And_get_the_binary_time;

 BEGIN
   The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
 END;

{ ------------------------------------------------------------- }

PROCEDURE Set_the_system_time;

 BEGIN
   SET_TIME     ( The_current_binary_time_is,
                  STATUS := Status_from_call_to_SET_TIME );

   IF NOT ODD   ( Status_from_call_to_SET_TIME ) THEN
        WRITELN (  'Call to SET_TIME failed with status: ',
                   Status_from_call_to_SET_TIME );
 END;

{ ------------------------------------------------------------- }

PROGRAM Set_system_time ( INPUT, OUTPUT );

VAR
        Status_returned : INTEGER;
        Jobs_port       : PORT;

BEGIN
        Prompt_user_for_time;
        And_get_the_binary_time;
        Set_the_system_time;

        INITIALIZATION_DONE ( STATUS := Status_returned );
           IF NOT ODD ( Status_returned ) THEN
              WRITELN ( 'Call to INITIALIZATION_DONE ',
                        ' failed with status: ', Status_returned );

END { of PROGRAM }.
END { of MODULE  };
```

D.3.1 Running TIME_2A

Use the command procedure TIME_2A.COM to build this system:

```
$       ! TIME_2A.COM
$       ! Command procedure to build the EPASCAL module TIME_2A
$       ! and the VAX FORTRAN program TODAY and build them into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_Time_2A:
$       EPASCAL -
                /LIST -
                /DEBUG TIME_2A
$ Compile_Today:
$       FORTRAN -
                /LIST -
                /NOOPTIMIZE -
                /DEBUG TODAY
$ Link_Time_2A:
$       LINK -
                /DEBUG TIME_2A -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ Link_Today:
$       LINK -
                /DEBUG -
                /NOSYSLIB TODAY -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT TIME_2A
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program TIME_2A /initialize /debug
program TODAY /debug
```

The difference here is the absence of the /norun qualification for TODAY

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.

```
{-----------------------------------------------------------------------
SOURCE:          TIME_3.PAS

PURPOSE:         To demonstrate request for time from adjacent VMS node
                 setting of that time on VAXELN and running of a
                 FORTRAN program in a separate job

                 APPLICATION11.PAS supplied with VAXELN shows the
                 mechanics of a request of VMS for a time string

COMPILE:         $ EPASCAL /LIST /DEBUG TIME_3

                 $ FORTRAN /NOOPTIMIZE /LIST /DEBUG TODAY

LINK:            $ LINK /DEBUG /NOSYSLIB TODAY, -
                 _$ ELN$:FRTLOBJECT /LIBRARY, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY

LINK:            $ LINK /DEBUG TIME_3, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT TIME_3

NEEDS:           TIME.COM in the default DECnet account.
                 TIME.COM looks like this

                 $ OPEN /WRITE Time_data SYS$NET
                 $ Current_time = F$TIME()
                 $ WRITE Time_data Current_time
                 $ CLOSE Time_data
                 $ EXIT

NOTES:           1) Command procedure TIME_3.COM compiles, links
                    and builds this module into a VAXELN system
-----------------------------------------------------------------------}
```

```
MODULE Get_VMS_time [IDENT ('V1.000')];

CONST
        LF                = ''(10);    { line-feed }
        Argument_position = 3;

VAR
        VAXELN_port,
        New_jobs_port   : PORT;
        VMS_message     : MESSAGE;
        Destination     : VARYING_STRING(15);
        VMS_time_string : ^STRING(32);
        Actual_time     : STRING(32);
        Current_time    : LARGE_INTEGER;
        Node_number     : VARYING_STRING (7);
        Status_returned : INTEGER;

{-------------------------------------------------}

PROGRAM Get_VMS_time;

BEGIN

{ find my job's port value instead of creating an additional
  port }

JOB_PORT ( VAXELN_port,
           STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'JOB_PORT status was: ', Status_returned :1 );

{ pick up the node number from the argument passed from the .DAT
  file at EBUILD and concatenate it with the object name TIME }

Node_number := PROGRAM_ARGUMENT ( Argument_position );
Destination := Node_number + '::TIME';

{ create the link with the VMS node whose number has just been
  collected }

WRITELN ( LF, '  1) About to connect with VMS node: ',
            Node_number, '...' );

CONNECT_CIRCUIT ( VAXELN_port,
                  DESTINATION_NAME := Destination,
                  STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'CONNECT_CIRCUIT status was: ', Status_returned :1

```

```
{ wait for VMS to respond with the time string }

WAIT_ANY ( VAXELN_port,
           STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'WAIT_ANY status was: ', Status_returned :1 );

RECEIVE ( VMS_message,
          VMS_time_string,
          VAXELN_port,
          STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'RECEIVE status was: ', Status_returned :1 );

DISCONNECT_CIRCUIT ( VAXELN_port,
                     STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'DISCONNECT_CIRCUIT status was: ',
                     Status_returned :1 );

WRITELN ( LF, '  2) Successful connection with VMS node' );

{ parse the received string for a 23 byte absolute time }

Actual_time  := SUBSTR ( VMS_time_string^, 1, 23 );
Current_time := TIME_VALUE ( Actual_time );

WRITELN ( LF, '  3) About to set system time...' );

SET_TIME ( Current_time,
           STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'SET_TIME status was: ',
                     Status_returned :1 );

WRITELN ( LF, '  4) Creating job to run VAX FORTRAN program', LF );

CREATE_JOB ( New_jobs_port,
             'TODAY',
             STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
           WRITELN ( 'CREATE_JOB status was: ', Status_returned :1 );

END {of PROGRAM}.
END {of MODULE };
```

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.

```
$       ! TIME_3.COM
$       ! Command procedure to build the EPASCAL module
$       ! TIME_3 and the VAX FORTRAN program TODAY and build them
$       ! into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_Time_3:
$       EPASCAL -
                /LIST -
                /DEBUG TIME_3
$ Compile_Today:
$       FORTRAN -
                /LIST -
                /NOOPTIMIZE -
                /DEBUG TODAY
$ Link_Time_3:
$       LINK -
                /DEBUG TIME_3 -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ Link_Today:
$       LINK -
                /DEBUG -
                /NOSYSLIB TODAY -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ !
$       SET NOVERIFY
$ Build_the_DAT_file:
$               @TIME_3_DAT
$       SET VERIFY
$ System_build:
$       EBUILD -
                        /NOEDIT TIME_3
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The command procedure TIME_3_DAT.COM looks like this:

```
$ !
$ !     MODULE:         TIME_3_DAT.COM
$ !     PURPOSE:        To build the .DAT file for TIME_3 automatically
$ !                     by grabbing the node area and node number using
$ !                     F$GETSYI the user can run the build without
$ !                     needing to know local node details
$ !
$ !     Finished file must look like this:
$ !
$ !     characteristic /noconsole
$ !     program TIME_3 /debug /argument=("""""","""""","""1.241""")
$ !     program TODAY /norun /debug
$ !
$       OPEN -
                /WRITE -
                Data_file -
                TIME_3.DAT
$ !
$ Set_up_symbols:
$       Area     = F$GETSYI ( "Node_area" )
$       Node_num = F$GETSYI ( "Node_number" )
$       Lits3    = """"""""
$       Lits6    = """"""""""""""
$       WRITE Data_file "characteristic /noconsole"
$       WRITE Data_file "program TIME_3 /debug /argument=(", -
        Lits6, ",", Lits6, ",", Lits3, area,".", node_num, Lits3, ")"
$       WRITE Data_file "program TODAY /norun /debug"
$
$ Close_file:
$       CLOSE Data_file
$       PURGE /NOLOG TIME_3.DAT
$       EXIT
```

The .DAT file used by EBUILD looked like this when EBUILD ran on node number 1.241

```
characteristic /noconsole
program TIME_3 /debug /argument=("""""","""""","""1.241""")
program TODAY /norun /debug
```

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

```
{-----------------------------------------------------------------------
SOURCE:         TIME_4.PAS

PURPOSE:        To demonstrate access of system symbols for VAXELN
                system time values.

COMPILE:        $ EPASCAL /LIST /DEBUG TIME_4

LINK:           $ LINK /DEBUG TIME_4, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT TIME_4

NOTES:          1) Command procedure TIME_4.COM compiles, links
                   and builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Time_4 [IDENT ('V1.000')];

CONST
        Nov_17_1858             = 0;    { earliest time }

VAR
        System_time_set         : BOOLEAN;

        Start_time,
        Idle_time,
        Previous_job_time       : LARGE_INTEGER;
```

```

        KER$GB_TIME_SET         : [EXTERNAL] BOOLEAN;

        KER$GQ_SYSTEM_TIME,
        KER$GQ_START_TIME,
        KER$GQ_IDLE_TIME,
        KER$GQ_PREV_JOB_TIME    : [EXTERNAL] LARGE_INTEGER;


{------------------ PROGRAM BLOCK ------------------}

PROGRAM Time_4 ( INPUT, OUTPUT );

BEGIN

{ Initialize variables...
  NOTE: KER$GQ_IDLE_TIME & KER$GQ_PREV_JOB_TIME are held as absolute
  times starting at 17-Nov-1858 00:00:00.00.

  When the time has been set:
        these must be subtracted from 17-Nov-1858 00:00:00.00
        to generate a delta time (bit 63 set)

  When the time is not set:
        the clock starts at 17-Nov-1858 00:00:00.00
}

System_time_set   := KER$GB_TIME_SET;
Start_time        := KER$GQ_START_TIME;

CASE System_time_set OF

  TRUE : BEGIN
          WRITELN ( 'System time has been set explicitly...' );
          Idle_time         := Nov_17_1858 - KER$GQ_IDLE_TIME;
          Previous_job_time := Nov_17_1858 - KER$GQ_PREV_JOB_TIME;
         END;

 FALSE : BEGIN
          Idle_time         := Start_time - KER$GQ_IDLE_TIME;
          Previous_job_time := Start_time - KER$GQ_PREV_JOB_TIME;
         END;
END {CASE};
```

```

        WRITELN ( 'System time: ',
                  TIME_STRING ( KER$GQ_SYSTEM_TIME ));
        WRITELN ( 'Start time:  ',
                  TIME_STRING ( Start_time ));
        WRITELN ( 'Idle time:         ',
                  TIME_STRING ( Idle_time ));
        WRITELN ( 'Previous job time: ',
                  TIME_STRING ( Previous_job_time ));

END {of PROGRAM}.
END {of MODULE};
```

D.5.1 Running TIME_4

Use the command procedure of the same name to build a system with TIME_4:

```
$       ! TIME_4.COM
$       ! Command procedure to compile and link the EPASCAL module
$       ! TIME_4
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile_Time_4:
$               @ELN_COMPILE_1 TIME_4
$ Link_Time_4:
$               @ELN_LINK_1 TIME_4
$ System_build:
$               @ELN_EBUILD_1 TIME_4
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file looks like this:

```
characteristic /noconsole /nonetwork /nofile /noserver
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.

```
{
SOURCE:         SYNCH_1.PAS

PURPOSE:        Demonstrates use of WAIT_ANY to delay for a period
                of seconds using the timeout facility of the WAIT_ANY
                procedure

COMPILE:        $ EPASCAL /LIST /DEBUG SYNCH_1

LINK:           $ LINK /DEBUG SYNCH_1, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SYNCH_1

NOTES:          1) Command procedure SYNCH_1.COM compiles, links and
                   builds this module into a system
_________________________________________________________________________
}
MODULE Synch_1 [IDENT ('V1.000')];

CONST
        Interval = '0000 00:00:05.00';     { 5 secs }

VAR
        Wait_result,
        Status_returned,
        Status_from_wait : INTEGER;

        Current_time,
        Binary_interval  : LARGE_INTEGER;

{ ------------------------------------------------------------ }
```

```

PROCEDURE Print_time;

BEGIN
    GET_TIME( Current_time,
              STATUS := Status_returned );
    IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'GET_TIME status was: ', Status_returned :1 );

    WRITELN ( 'The time now is: ', TIME_STRING (Current_time) );
END;

{ ------------------------------------------------------------ }

PROGRAM Synch_1 (OUTPUT);

BEGIN
    Binary_interval := TIME_VALUE ( Interval );

    Print_time;
    WRITELN ( 'About to wait...... ' );

    WAIT_ANY (  RESULT := Wait_result,
                TIME   := Binary_interval,
                STATUS := Status_from_wait );
    IF NOT ODD ( Status_from_wait ) THEN
        WRITELN ( 'WAIT status was: ', Status_from_wait:1);

    WRITELN ( 'Wait completed' );
    WRITE   ( 'Value of wait_result was: ', Wait_result:1 );

    IF Wait_result = 0 THEN
        WRITELN ( ' indicating timeout.' )
    ELSE
        WRITELN;

    Print_time;

END { of PROGRAM }.
END { of MODULE  };
```

D.6.1 Running SYNCH_1

Use the command procedure of the same name to build a system with SYNCH_1:

```
$       ! SYNCH_1.COM
$       ! Command procedure to build the VAXELN
$       ! module SYNCH_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 SYNCH_1
$ Link:
$               @ELN_LINK_1 SYNCH_1
$ System_build:
$               @ELN_EBUILD_1 SYNCH_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
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
```

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

```
{
SOURCE:         SYNCH_2.PAS

PURPOSE:        Demonstrates waiting for an event

COMPILE:        $ EPASCAL /LIST /DEBUG SYNCH_2

LINK:           $ LINK /DEBUG SYNCH_2, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SYNCH_2

NOTES:          1) Command procedure SYNCH_2.COM compiles, links and
                   builds this module into a system
____________________________________________________________________
}
MODULE Synch_2 [IDENT ('V1.000')];

CONST
        Interval = '0000 00:00:05.00';     { 5 secs }

VAR
        Process_ID       : PROCESS;

        Binary_interval  : LARGE_INTEGER;

        Wait_result,
        Status_returned,
        Status_from_signal,
        Status_from_wait : INTEGER;


{ ------------------------------------------------------------- }

```

```
PROCESS_BLOCK Flag_setter ( Flag_to_signal : EVENT );

BEGIN
   Binary_interval := TIME_VALUE ( Interval );

 WAIT_ANY (      RESULT := Wait_result,
                 TIME   := Binary_interval,
                 STATUS := Status_from_wait );
   IF NOT ODD ( Status_from_wait ) THEN
       WRITELN ( 'WAIT status was: ', Status_from_wait:1);

   WRITELN ( 'In subprocess about to signal event... ');

   SIGNAL  ( Flag_to_signal,
               STATUS := Status_from_signal );
   IF NOT ODD ( Status_from_signal ) THEN
       WRITELN ( 'SIGNAL status was: ', Status_from_signal:1 );

END;
{ -------------------------------------------------------------- }
PROGRAM Synch_2 (OUTPUT);

VAR
       Event_flag       : EVENT;

BEGIN
 CREATE_EVENT (     Event_flag,
                    EVENT$CLEARED,
                    STATUS := Status_returned );
   IF NOT ODD ( Status_returned ) THEN
       WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);

 CREATE_PROCESS (   Process_id,
                    Flag_setter,
                    Event_flag,
                    STATUS := Status_returned );
   IF NOT ODD ( Status_returned ) THEN
       WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);

   WRITELN ( 'Master process about to wait...... ' );

 WAIT_ANY      (    Event_flag,
                    RESULT := Wait_result,
                    STATUS := Status_from_wait );
   IF NOT ODD ( Status_from_wait ) THEN
       WRITELN ( 'WAIT status was: ', Status_from_wait:1);

   WRITELN ( 'Wait completed' );
   WRITE   ( 'Value of wait_result was: ', Wait_result:1 );

END { of PROGRAM }.
END { of MODULE  };
```

D.7.1 Running SYNCH_2

Use the command procedure of the same name to build a system with SYNCH_2:

```
$       ! SYNCH_2.COM
$       ! Command procedure to build the VAXELN
$       ! module SYNCH_2
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 SYNCH_2
$ Link:
$               @ELN_LINK_1 SYNCH_2
$ System_build:
$               @ELN_EBUILD_1 SYNCH_2
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
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
```

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

```
{
SOURCE:          SYNCH_3.PAS

PURPOSE:         Demonstrates waiting for several objects
                 Three of the objects are dummies just for
                 demonstration purposes.

COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_3

LINK:            $ LINK /DEBUG SYNCH_3, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT SYNCH_3

NOTES:           1) Command procedure SYNCH_3.COM compiles, links and
                    builds this module into a system

}
MODULE Synch_3 [IDENT ('V1.000')];

CONST
        Interval          = '0000 00:00:05.00';    { 5 secs }
        No_of_dummy_events = 3;

VAR
        Process_ID        : PROCESS;
        Dummy_event       : ARRAY [1..No_of_dummy_events] OF EVENT;
        Event_flag        : EVENT;

        Binary_interval   : LARGE_INTEGER;

        Wait_result,
        Status_returned,
        Status_from_signal,
        Status_from_wait : INTEGER;


{ ------------------------------------------------------------ }
```

```

PROCESS_BLOCK Flag_setter ( Flag_to_signal : EVENT );

BEGIN
    Binary_interval := TIME_VALUE ( Interval );

{ wait for some seconds before signalling the event }

 WAIT_ANY (      RESULT := Wait_result,
                 TIME   := Binary_interval,
                 STATUS := Status_from_wait );
    IF NOT ODD ( Status_from_wait ) THEN
        WRITELN ( 'WAIT status was: ', Status_from_wait:1);

    WRITELN ( 'In subprocess about to signal event... ');

    SIGNAL  ( Flag_to_signal,
              STATUS := Status_from_signal );
    IF NOT ODD ( Status_from_signal ) THEN
        WRITELN ( 'SIGNAL status was: ', Status_from_signal:1 );

END;

{ ------------------------------------------------------------- }

PROGRAM Synch_3 (OUTPUT);

VAR
     Event_count : 1..No_of_dummy_events;

BEGIN
 CREATE_EVENT (    Event_flag,
                   EVENT$CLEARED,
                   STATUS := Status_returned );
    IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);

{ create three dummy objects - in this case EVENT objects }

 FOR Event_count := 1 TO No_of_dummy_events DO
  BEGIN
        CREATE_EVENT ( Dummy_event[Event_count],
                       EVENT$CLEARED,
                       STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
          WRITELN ( 'CREATE_EVENT status was: ', Status_returned :1);
  END;

```

```
 CREATE_PROCESS (  Process_id,
                   Flag_setter,
                   Event_flag,
                   STATUS := Status_returned );
    IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);

    WRITELN ( 'Master process about to wait...... ' );

{ wait for any object to be signalled }

 WAIT_ANY       (   Dummy_event[1],
                    Dummy_event[2],
                    Event_flag,
                    Dummy_event[3],
                    RESULT := Wait_result,
                    STATUS := Status_from_wait );
    IF NOT ODD ( Status_from_wait ) THEN
        WRITELN ( 'WAIT status was: ', Status_from_wait:1);

    WRITELN ( 'Wait completed' );
    WRITE   ( 'Value of wait_result was: ', Wait_result:1 );

END { of PROGRAM }.
END { of MODULE  };
```

D.8.1 Running SYNCH_3

Use the command procedure of the same name to build a system with SYNCH_3:

```
$       ! SYNCH_3.COM
$       ! Command procedure to build the VAXELN
$       ! module SYNCH_3
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 SYNCH_3
$ Link:
$               @ELN_LINK_1 SYNCH_3
$ System_build:
$               @ELN_EBUILD_1 SYNCH_3
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
program SYNCH_3 /debug
```

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.

```
{
SOURCE:         SYNCH_4.PAS

PURPOSE:        Demonstrates waiting for a process object

COMPILE:        $ EPASCAL /LIST /DEBUG SYNCH_4

LINK:           $ LINK /DEBUG SYNCH_4, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SYNCH_4

NOTES:          1) Command procedure SYNCH_4.COM compiles, links and
                   builds this module into a system
______________________________________________________________________
}
MODULE Synch_4 [IDENT ('V1.000')];

VAR
        Event_flag        : EVENT;
        Subprocess_id     : PROCESS;

        Wait_result,
        Status_returned,
        Status_from_wait : INTEGER;

{ ------------------------------------------------------------- }

PROCESS_BLOCK Time_waster ( Flag : EVENT );

VAR
   I,
   J,
   Ret_stat      : INTEGER;

BEGIN

```

```
{  process waits here for the master process to signal the event thus
   satisfying the wait and setting the subprocess on its way }

   WAIT_ANY (   Flag,
                STATUS := Ret_stat );
    IF NOT ODD ( Ret_stat ) THEN
       WRITELN ( 'WAIT status was: ', Ret_stat:1 );

{ just waste some time }

   FOR I := 1 TO 10000 DO
     J := I;
END;

{ ------------------------------------------------------------- }

PROGRAM Synch_4 (OUTPUT);

BEGIN

{ create the event that will hold the subprocess after its created }

  CREATE_EVENT   (       Event_flag,
                         EVENT$CLEARED,
                         STATUS := Status_returned );
         IF NOT ODD ( Status_returned ) THEN
           WRITELN  ( 'CREATE_EVENT status was: ', Status_returned :1);

  CREATE_PROCESS (       Subprocess_id,
                         Time_waster,
                         Event_flag,
                         STATUS := Status_returned );
        IF NOT ODD  ( Status_returned ) THEN
        WRITELN   ( 'CREATE_PROCESS status was: ', Status_returned :1);

  WRITELN ( 'Master process about to wait...... ' );

{ let the subprocess go by 'dropping' the flag }

  SIGNAL ( Event_flag,
           STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
        WRITELN    ( 'SIGNAL status was: ', Status_returned :1);
```

```

{ wait for the subprocess to expire }

  WAIT_ANY       (  Subprocess_id,
                    RESULT := Wait_result,
                    STATUS := Status_from_wait );
    IF NOT ODD ( Status_from_wait ) THEN
        WRITELN ( 'WAIT status was: ', Status_from_wait:1);

    WRITELN ( 'Wait completed' );
    WRITE   ( 'Value of wait_result was: ', Wait_result:1 );

END { of PROGRAM }.
END { of MODULE  };
```

D.9.1 Running SYNCH_4

Use the command procedure of the same name to build a system with SYNCH_4:

```
$        ! SYNCH_4.COM
$        ! Command procedure to build the VAXELN
$        ! module SYNCH_4
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 SYNCH_4
$ Link:
$               @ELN_LINK_1 SYNCH_4
$ System_build:
$               @ELN_EBUILD_1 SYNCH_4
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
program SYNCH_4 /debug
```

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.

```
{
SOURCE:          SYNCH_5.PAS

PURPOSE:         Demonstrates use of a MUTEX to control access to an
                 array

COMPILE:         $ EPASCAL /LIST /DEBUG SYNCH_5, ELN$:RTLOBJECT /LIB

LINK:            $ LINK /DEBUG SYNCH_5, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT SYNCH_5

NOTES:           1) Command procedure SYNCH_5.COM compiles, links and
                    builds this module into a system

}
MODULE Synch_5 [IDENT ('V1.000')];

INCLUDE
        $MUTEX;         { contains mutex definitions }

CONST
        LF         = ''(10);    { line-feed }
        Fill_limit = 5;
        Limit      = 100;
VAR
        List_of_integers : ARRAY [1..Limit] OF INTEGER;
        Access_control   : MUTEX;
        Number_of_fills  : INTEGER := ZERO;
        Status_returned  : INTEGER;

{ ------------------------------------------------------------ }
```

```

PROCESS_BLOCK Array_filler;

VAR
        I : INTEGER;

BEGIN
 WHILE Number_of_fills < Fill_limit DO
  BEGIN

{ lock the mutex created by the master process - this increments
  the count for the mutex. When the master process 'unlocks', the
  count is decremented and at most one process - this one - can
  proceed
}
   LOCK_MUTEX ( Access_control );

   FOR I := 1 TO Limit DO List_of_integers[I] :=
                          List_of_integers[I] + I * 2;

   Number_of_fills := Number_of_fills + 1;

   WRITELN ( LF, 'Fill number ', Number_of_fills:1,
                 ' completed, contents of array --> ', LF);

{ now let the next waiting process gain control of the mutex
  - 'Array_writer' }

   UNLOCK_MUTEX ( Access_control );
  END;
END;

{ ---------------------------------------------------------------- }

PROCESS_BLOCK Array_writer;

VAR
        I  : 1..Limit;

BEGIN
 WHILE Number_of_fills < Fill_limit DO
  BEGIN

{ stop other processes using the array by locking the mutex }

   LOCK_MUTEX ( Access_control );
       FOR I := 1 TO Limit DO
         BEGIN
           IF    ( (I-1) MOD 16 = 0 ) THEN WRITELN;
           WRITE ( List_of_integers[I]:5 );
         END;
        WRITELN;

```

```
{ decrement the mutex counter - allow the next waiting process,
  if any, to continue }

   UNLOCK_MUTEX ( Access_control );
  END;
END;

{ ------------------------------------------------------------- }

PROGRAM Synch_5 (OUTPUT);

VAR
        Filler_ID,
        Writer_ID       : PROCESS;

BEGIN

{ creating the mutex initializes its counter to -1 }

CREATE_MUTEX ( Access_control,
               Status_returned );
         IF NOT ODD ( Status_returned ) THEN
           WRITELN  ( 'CREATE_MUTEX status was: ', Status_returned :1);

{ this master process now has control of the mutex and increments
  the mutex counter }

LOCK_MUTEX ( Access_control );

{ create two processes that will manipulate the array }

CREATE_PROCESS ( Filler_id,
                 Array_filler,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN  ( 'CREATE_PROCESS status was: ', Status_returned :1);

CREATE_PROCESS ( Writer_id,
                 Array_writer,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN  ( 'CREATE_PROCESS status was: ', Status_returned :1);

{ This ensures we don't miss the first or second write operations }

         WRITELN ( 'Please hit the return key to start' );
         READLN;
```

```
{ let the first process go - Array_filler }

UNLOCK_MUTEX ( Access_control );

{ wait for the two processes to complete their operations and exit }

WAIT_ALL ( Filler_ID,
           Writer_ID,
           STATUS := Status_returned );
         IF NOT ODD ( Status_returned ) THEN
         WRITELN ( 'WAIT status was: ', Status_returned:1);

END { of PROGRAM }.
END { of MODULE  };
```

D.10.1 Running SYNCH_5

Use the command procedure of the same name to build a system with SYNCH_5:

```
$       ! SYNCH_5.COM
$       ! Command procedure to compile and link the VAXELN
$       ! module SYNCH_5
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG SYNCH_5, ELN$:RTLOBJECT /LIBRARY
$       SET NOVERIFY
$ Link:
$               @ELN_LINK_1 SYNCH_5
$ System_build:
$               @ELN_EBUILD_1 SYNCH_5
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
program SYNCH_5 /debug
```

If, when using EDEBUG, 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.

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
```

```
!
!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
```

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

```
{
SOURCE:         SYNCH_6.PAS

PURPOSE:        Demonstrates multiple processes synchronizing
                using a mutex

COMPILE:        $ EPASCAL /LIST /DEBUG SYNCH_6

LINK:           $ LINK /DEBUG SYNCH_6, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT SYNCH_6

NOTES:          1) Command procedure SYNCH_6.COM compiles, links and
                   builds this module into a system
-----------------------------------------------------------------------
}
MODULE Synch_6 [IDENT ('V1.000')];

INCLUDE
        $MUTEX;

CONST
        Process_limit   = 100;

VAR
        Process_IDs     : ARRAY [1..Process_limit] OF PROCESS;
        Process_lock    : MUTEX;
        Status_returned : INTEGER;

{ ------------------------------------------------------------ }

```

```
PROCEDURE Write_time_now;

VAR
        Now : LARGE_INTEGER;

BEGIN
        GET_TIME ( Now );    { no status checks for speed }
        WRITELN  ( TIME_STRING ( Now ) );
END;

{ ----------------------------------------------------------- }

PROCESS_BLOCK Subprocess ( K : INTEGER );

BEGIN

LOCK_MUTEX ( Process_lock );

  WRITE ( K : 4 );

  IF ( K MOD 20 = 0 ) THEN WRITELN;

UNLOCK_MUTEX ( Process_lock );

END;

{ ----------------------------------------------------------- }

PROGRAM Synch_6 ( OUTPUT );

VAR
        I               : 1..Process_limit;

BEGIN

CREATE_MUTEX ( Process_lock );
LOCK_MUTEX   ( Process_lock );

WRITE ( 'About to create ', Process_limit:1, ' processes... ');
Write_time_now;

FOR I := 1 TO Process_limit DO
   BEGIN
        CREATE_PROCESS ( Process_IDs [I],
                         Subprocess,
                         I,
                         STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
                WRITELN ( 'Bad status returned by CREATE_PROCESS: ',
                          Status_returned:1 );
   END;
```

```
WRITE ( '    ...process creation complete ');
Write_time_now;

WRITELN ( 'Please hit return to start... ');
READLN;

WRITELN ( ' ':10, '------------- SUBPROCESS OUTPUT -------------' );
WRITELN;

UNLOCK_MUTEX ( Process_lock );

WAIT_ANY ( Process_IDs [Process_limit] );

END {of PROGRAM}.
END {of MODULE};
```

D.11.1 Running SYNCH_6

Use the command procedure of the same name to build a system with SYNCH_6:

```
$        ! SYNCH_6.COM
$        ! Command procedure to compile and link the VAXELN
$        ! module SYNCH_6
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ Compile:
$        EPASCAL -
                 /LIST -
                 /DEBUG SYNCH_6, ELN$:RTLOBJECT /LIBRARY
$        SET NOVERIFY
$ Link:
$                @ELN_LINK_1 SYNCH_6
$ System_build:
$                @ELN_EBUILD_1 SYNCH_6
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT files looks like this:

```
characteristic /noconsole /nonetwork /nofile /noserver -
        /objects=800 /processes=120
program SYNCH_6 /debug
```

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.

```
{
SOURCE:         COMM_1.PAS

PURPOSE:        Demonstrates communication between a subprocess and
                a master process using a datagram

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_1

LINK:           $ LINK /DEBUG COMM_1, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_1

NOTES:          1) Command procedure COMM_1.COM compiles, links and
                   builds this module into a system
_____________________________________________________________________
}
```

```
MODULE Comm_1 [IDENT ('V1.000')];

{ define a message 'packet' }

TYPE
        Message_packet = RECORD
            Message_number : 0..MAXINT;
            Timestring     : VARYING_STRING(23);
        END;

VAR
        Datagram         : ^Message_packet;
        Jobs_port_value  : PORT;
        Message_ID       : MESSAGE;
        Status_returned  : INTEGER;
        Sub_ID           : PROCESS;

{ ------------------------------------------------------------ }

PROCESS_BLOCK Sender;

VAR
        Current_time_binary : LARGE_INTEGER;

BEGIN

GET_TIME ( Current_time_binary,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'GET_TIME status was: ',
                       Status_returned :1);

{ complete the datagram's contents... }

WITH Datagram^ DO
  BEGIN
    Message_number := Datagram^.Message_number + 1;
    Timestring     := TIME_STRING ( Current_time_binary );
  END;

{ ... then send it }

SEND ( Message_ID,
       Jobs_port_value,
       STATUS := Status_returned );
    IF NOT ODD  ( Status_returned ) THEN
    WRITELN     ( 'SEND status was: ',
                       Status_returned :1);
END;

{ ------------------------------------------------------------ }
```

```
PROGRAM Comm_1 (OUTPUT);

BEGIN

{ get a pointer to a datagram packet }

CREATE_MESSAGE ( Message_ID,
                 Datagram,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_MESSAGE status was: ',
                        Status_returned :1);

Datagram^.Message_number := 0;

{ find the port value for the master process default port }

JOB_PORT ( Jobs_port_value,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'JOB_PORT status was: ',
                        Status_returned :1);
{ create the subprocess that's going to communicate with the
  datagram
}

CREATE_PROCESS ( Sub_ID,
                 Sender,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_PROCESS status was: ',
                        Status_returned :1);

{ wait on the port i.e. wait for a message to arrive }

WAIT_ANY ( Jobs_port_value,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                        Status_returned :1);

RECEIVE ( Message_ID,
          Datagram,
          Jobs_port_value,
          STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'RECEIVE status was: ',
                        Status_returned :1);
```

```
{ read the message and display its contents }

WITH Datagram^ DO
  BEGIN
        WRITELN ( 'Contents of the message received: ' );
        WRITELN ( 'Message number: ', Message_number:1 );
        WRITELN ( 'Message text:   ', Timestring );
  END;

END { of PROGRAM }.
END { of MODULE  };
```

E.1.1 Running COMM_1

Use the command procedure of the same name to build a system with COMM_1:

```
$       ! COMM_1.COM
$       ! Command procedure to build the VAXELN
$       ! module COMM_1
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 COMM_1
$ Link:
$               @ELN_LINK_1    COMM_1
$ System_build:
$               @ELN_EBUILD_1   COMM_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
program COMM_1 /debug
```

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.

```
{
SOURCE:         COMM_2.PAS

PURPOSE:        Demonstrates communication between a subprocess and
                a master process using a datagram.
                This is an enhancement of COMM_1 and involves
                communication in two directions

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_2

LINK:           $ LINK /DEBUG COMM_2, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_2

NOTES:          1) Command procedure COMM_2.COM compiles, links and
                   builds this module into a system
______________________________________________________________________
}
MODULE Comm_2 [IDENT ('V1.000')];

CONST
        LF      = ''(10);       { line-feed }

{ the message packet (datagram) definition }

TYPE
        Message_packet = RECORD
            Message_number : 0..MAXINT;
            Timestring     : VARYING_STRING(23);
        END;

VAR
        Datagram         : ^Message_packet;
        Green_light      : EVENT;
        Jobs_port_value  : PORT;
        Message_ID       : MESSAGE;
        Status_returned  : INTEGER;
        Sub_ID           : PROCESS;

{ ------------------------------------------------------------ }
```

```

PROCEDURE Send_message;

BEGIN

SEND ( Message_ID,
       Jobs_port_value,
       STATUS := Status_returned );
    IF NOT ODD  ( Status_returned ) THEN
    WRITELN     ( 'SEND status was: ',
                                Status_returned :1);

END;

{ ------------------------------------------------------------- }

PROCEDURE Receive_message_and_display_it;

BEGIN

RECEIVE ( Message_ID,
          Datagram,
          Jobs_port_value,
          STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'RECEIVE status was: ',
                              Status_returned :1);

WITH Datagram^ DO
  BEGIN
        WRITELN ( 'Contents of the message received: ' );
        WRITELN ( 'Message number: ', Message_number:1 );
        WRITELN ( 'Message text:   ', Timestring );
  END;

END;

{ ------------------------------------------------------------- }
```

```

PROCESS_BLOCK Sender;

VAR
        Current_time_binary : LARGE_INTEGER;

BEGIN

GET_TIME ( Current_time_binary,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'GET_TIME status was: ',
                        Status_returned :1);

WITH Datagram^ DO
  BEGIN
    Message_number := Datagram^.Message_number + 1;
    Timestring     := TIME_STRING ( Current_time_binary );
  END;

WRITELN ( 'Subprocess about to send...', LF );

Send_message;

WAIT_ALL ( Jobs_port_value,
           Green_light,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                        Status_returned :1);

WRITELN ( 'Subprocess about to receive...', LF );

Receive_message_and_display_it;

END;

{ ----------------------------------------------------------------- }
```

```


PROGRAM Comm_2 (OUTPUT);

BEGIN

{ get a pointer to the message packet }

CREATE_MESSAGE ( Message_ID,
                 Datagram,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_MESSAGE status was: ',
                        Status_returned :1);

Datagram^.Message_number := 0;

JOB_PORT ( Jobs_port_value,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'JOB_PORT status was: ',
                        Status_returned :1);

{ event for holding the subprocess }

CREATE_EVENT ( Green_light,
               EVENT$CLEARED,
               STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_EVENT status was: ',
                        Status_returned :1);

CREATE_PROCESS ( Sub_ID,
                 Sender,
                 STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_PROCESS status was: ',
                        Status_returned :1);

```

```
{ hang around for the message to arrive }

WAIT_ANY ( Jobs_port_value,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                          Status_returned :1);

WRITELN ( 'Master process about to receive...', LF );

Receive_message_and_display_it;

{ update the message packet }

WITH Datagram^ DO
  BEGIN
   Message_number := Datagram^.Message_number + 1;
   Timestring     := 'Closing message ...';
  END;

WRITELN ( LF, 'Master process about to send...', LF );

Send_message;

{ give the subprocess the green light to continue }

SIGNAL ( Green_light,
         STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'SIGNAL status was: ',
                          Status_returned :1);

{ wait for the subprocess to die }

WAIT_ANY ( Sub_ID,
           STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                          Status_returned :1);

END { of PROGRAM }.
END { of MODULE  };
```

E.2.1 Running COMM_2

Use the command procedure of the same name to build a system with COMM_2:

```
$        ! COMM_2.COM
$        ! Command procedure to build the VAXELN module COMM_2
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$ Compile:
$                @ELN_COMPILE_1 COMM_2
$ Link:
$                @ELN_LINK_1    COMM_2
$ System_build:
$                @ELN_EBUILD_1   COMM_2
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile /noserver
program COMM_2 /debug
```

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
```

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.

```
{
SOURCE:         COMM_3.PAS

PURPOSE:        Demonstrates communication between jobs using the
                datagram method.
                This program formats a message that includes an array
                of random numbers generated by a VAX FORTRAN function

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_3

LINK:           $ LINK /DEBUG COMM_3, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_3

NOTES:          1) Command procedure COMM_3.COM compiles, links and
                   builds this module into a system

}
MODULE Comm_3 [IDENT ('V1.000')];

{ the external FORTRAN routine - note the attribute REFERENCE -
  FORTRAN expects to receive the address of numeric data
  in a call to one of its FUNCTIONs or SUBROUTINEs
}

FUNCTION Random ( Seed : [REFERENCE] INTEGER ): REAL; EXTERNAL;

CONST
        Limit = 100;

{ define the message packet }

TYPE
        Message_packet = RECORD
            Message_number : 0..MAXINT;
            Binary_time    : LARGE_INTEGER;
            Some_numbers   : ARRAY[1..Limit] OF INTEGER;
        END;
```

```

VAR
        Other_port,
        Jobs_port_value : PORT;
        Telegram        : ^Message_packet;
        Message_ID      : MESSAGE;
        Name_ID         : NAME;
        Status_returned : INTEGER;

{ ------------------------------------------------------------- }

PROGRAM Comm_3 (OUTPUT);

VAR
        I               : INTEGER;
        Randoms_seed    : INTEGER := 1987;  { odd numbers are
                                              preferred seed values }

BEGIN

{ get a pointer to the message packet }

CREATE_MESSAGE ( Message_ID,
                 Telegram,
                 STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_MESSAGE status was: ',
                        Status_returned :1);

JOB_PORT ( Jobs_port_value,
           STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'JOB_PORT status was: ',
                        Status_returned :1);

{ find the value of the other port - we only know its name so pass that
  via the kernel to the name server. That will do the translation for
  us.
}

TRANSLATE_NAME ( Other_port,
                 'NEW YORK',
                 NAME$UNIVERSAL,
                 STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'TRANSLATE_NAME status was: ',
                        Status_returned :1);

```

```
{ load the message packet }

WITH Telegram^ DO
  BEGIN
    Message_number := 1;
    GET_TIME ( Binary_time,
               STATUS := Status_returned );
         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                        Status_returned :1);

    FOR I := 1 TO Limit DO
        Some_numbers[I] := TRUNC ( (Random (Randoms_seed) * 1000.0) );
  END;

{ dispatch the packet to 'NEW YORK' }

SEND ( Message_ID,
       Other_port,
       STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'SEND status was: ',
                        Status_returned :1);

END { of PROGRAM }.
END { of MODULE  };
```

The other job in this system runs the program COMM_3A.PAS

```
{
SOURCE:         COMM_3A.PAS

PURPOSE:        Demonstrates communication between jobs.
                This program runs before COMM_3 to establish a name
                for its job port then signals "INITIALIZATION DONE"

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_3A

LINK:           $ LINK /DEBUG COMM_3A, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_3

NOTES:          1) Command procedure COMM_3.COM compiles, links and
                   builds this module into a system
______________________________________________________________________
}
MODULE Comm_3A [IDENT ('V1.000')];

CONST
        Limit = 50;

TYPE
        Message_packet = RECORD
            Message_number : 0..MAXINT;
            Binary_time    : LARGE_INTEGER;
            Some_numbers   : ARRAY[1..Limit] OF INTEGER;
        END;

VAR
        Jobs_port_value : PORT;
        Telegram        : ^Message_packet;
        Message_ID      : MESSAGE;
        Name_ID         : NAME;
        Status_returned : INTEGER;

{ ---------------------------------------------------------------- }

PROGRAM Comm_3A (OUTPUT);

VAR
        I               : INTEGER;

```

```
BEGIN

CREATE_MESSAGE ( Message_ID,
                 Telegram,
                 STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_MESSAGE status was: ',
                        Status_returned :1);

JOB_PORT ( Jobs_port_value,
           STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'JOB_PORT status was: ',
                        Status_returned :1);

{ putting the port's name in the UNIVERSAL name table means the job
  could run on a separate VAXELN node without modification. Datagrams
  are not reliable especially over that sort of distance
}

CREATE_NAME ( Name_ID,
              'NEW YORK',
              Jobs_port_value,
              TABLE  := NAME$UNIVERSAL,
              STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_NAME status was: ',
                        Status_returned :1);

{ allow the other job running COMM_3 to start now }

INITIALIZATION_DONE ( STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'INITIALIZATION_DONE status was: ',
                        Status_returned :1);
```

```
{ wait for the message to arrive }

WAIT_ANY ( Jobs_port_value,
           STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                        Status_returned :1);

RECEIVE ( Message_ID,
          Telegram,
          Jobs_port_value,
          STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'SEND status was: ',
                        Status_returned :1);

{ display the message }

WITH Telegram^ DO
  BEGIN
    WRITELN ( 'Message number: ', Message_number:1 );
    WRITELN ( 'Time of message: ', TIME_STRING ( Binary_time ) );
    FOR I := 1 TO Limit DO
     BEGIN
        IF ( (I-1) MOD 10 = 0 ) THEN WRITELN;
        WRITE ( Some_numbers[I]: 6 );
     END;
    WRITELN;
  END;

END { of PROGRAM }.
END { of MODULE  };
```

The FORTRAN function looks like this:

```
*
*       SOURCE:         RANDOM.FOR
*       USES:           VAX FORTRAN built-in RAN function
*
        REAL FUNCTION Random ( Seed )

        IMPLICIT NONE

        INTEGER         Seed
        REAL            Num

        Random = RAN ( Seed )

        RETURN
        END
```

E.3.1 Running COMM_3

Use the command procedure of the same name to build a system with COMM_3:

```
$       ! COMM_3.COM
$       ! Command procedure to build the VAXELN module COMM_3
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 COMM_3
$               @ELN_COMPILE_1 COMM_3A
$       SET VERIFY
$       FORTRAN -
                /NOOPTIMIZE -
                /DEBUG -
                /LIST RANDOM
$ Link:
$       LINK -
                /NOSYSLIB -
                /DEBUG  COMM_3, RANDOM -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,RTLSHARE /LIBRARY -
                ,RTL /LIBRARY
$       SET NOVERIFY
$               @ELN_LINK_1    COMM_3A
$ System_build:
$               @ELN_EBUILD_1   COMM_3
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program COMM_3 /debug
program COMM_3A /initialize /debug
```

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.

```
{
SOURCE:         COMM_4.PAS

PURPOSE:        Demonstrates communication between jobs using
                datagrams.
                This program is a variation on COMM_3A and gets the
                port name from the program's arguments.

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_4

LINK:           $ LINK /DEBUG COMM_4, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_4

NOTES:          1) Command procedure COMM_4.COM compiles, links and
                   builds this module into a system
________________________________________________________________________
}
MODULE Comm_4 [IDENT ('V1.000')];

CONST
        Limit = 50;

TYPE
        Message_packet = RECORD
            Message_number : 0..MAXINT;
            Binary_time    : LARGE_INTEGER;
            Some_numbers   : ARRAY[1..Limit] OF INTEGER;
        END;

VAR
        Argument_number : INTEGER := 3;
        Jobs_port_value : PORT;
        Telegram        : ^Message_packet;
        Message_ID      : MESSAGE;
        Name_ID         : NAME;
        Status_returned : INTEGER;
        Port_name_arg   : VARYING_STRING(32);

{ ---------------------------------------------------------- }
```

```

PROGRAM Comm_4 (OUTPUT);

VAR
        I                  : INTEGER;

BEGIN

CREATE_MESSAGE ( Message_ID,
                 Telegram,
                 STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_MESSAGE status was: ',
                        Status_returned :1);

JOB_PORT ( Jobs_port_value,
           STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'JOB_PORT status was: ',
                        Status_returned :1);

Port_name_arg := PROGRAM_ARGUMENT ( Argument_number );

CREATE_NAME ( Name_ID,
              Port_name_arg,
              Jobs_port_value,
              TABLE  := NAME$UNIVERSAL,
              STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'CREATE_NAME status was: ',
                        Status_returned :1);

INITIALIZATION_DONE ( STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'INITIALIZATION_DONE status was: ',
                        Status_returned :1);

WAIT_ANY ( Jobs_port_value,
           STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'WAIT status was: ',
                        Status_returned :1);
```

```

RECEIVE ( Message_ID,
          Telegram,
          Jobs_port_value,
          STATUS := Status_returned );

         IF NOT ODD  ( Status_returned ) THEN
         WRITELN     ( 'SEND status was: ',
                        Status_returned :1);

WITH Telegram^ DO
  BEGIN
    WRITELN ( 'Message number: ', Message_number:1 );
    WRITELN ( 'Time of message: ', TIME_STRING ( Binary_time ) );
    FOR I := 1 TO Limit DO
     BEGIN
        IF ( (I-1) MOD 10 = 0 ) THEN WRITELN;
        WRITE ( Some_numbers[I]: 6 );
     END;
    WRITELN;
  END;

END { of PROGRAM }.
END { of MODULE  };
```

E.4.1 Running COMM_4

Use the command procedure of the same name to build a system with COMM_4:

```
$       ! COMM_4.COM
$       ! Command procedure to build the VAXELN module COMM_4
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$ Compile:
$               @ELN_COMPILE_1 COMM_3
$               @ELN_COMPILE_1 COMM_4
$       SET VERIFY
$       FORTRAN -
                /NOOPTIMIZE -
                /DEBUG -
                /LIST RANDOM
$ Link:
$       LINK -
                /NOSYSLIB -
                /DEBUG  COMM_3, RANDOM -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,RTLSHARE /LIBRARY -
                ,RTL /LIBRARY
$       SET NOVERIFY
$               @ELN_LINK_1    COMM_4
$ System_build:
$               @ELN_EBUILD_1   COMM_4
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program COMM_3 /debug
program COMM_4 /initialize /debug /argument=("""""", """""", """NEW YORK""")
```

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
```

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.

```
{----------------------------------------------------------------------
SOURCE:         COMM_5.PAS

PURPOSE:        To demonstrate creation of, and access to,
                a VAXELN AREA

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_5, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG COMM_5, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT COMM_5

NOTES:          1) Command procedure COMM_5.COM compiles, links and
                   builds this module into a VAXELN system
----------------------------------------------------------------------}
MODULE Create_area_01 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        Master_process_greeting = 'This is from the master process';
        Process_greeting        = 'This is from the created process';
        Limit                   = 100;

TYPE
        List_of_integers        = ARRAY [1..Limit] OF INTEGER;

VAR
        Area_identity           : AREA;
        Process_identity        : PROCESS;
        Start_of_area           : ^List_of_integers;
        Returned_status         : INTEGER;

{--------------------------------------------------}
```

```
PROCEDURE Write_contents_of_area;

VAR
        I                  : 1..Limit;

BEGIN
    FOR I := 1 TO Limit DO
      BEGIN
        IF      ( (I-1) MOD 8 = 0)
             THEN WRITELN;
        WRITE   ( Start_of_area^ [I] : 10 );
      END;
    WRITELN;
END;

{---------------------------------------------------}

PROCEDURE Create_process_and_wait;

BEGIN
CREATE_PROCESS ( Process_identity,
                 Area_accessor,
                 STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_PROCESS' );

WAIT_ANY ( Process_identity,
           STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );
END;

{---------------------------------------------------}
```

```

PROCEDURE Issue_master_process_greeting;

BEGIN
   WRITELN ( Master_process_greeting );
END;

{---------------------------------------------------}

PROCEDURE Create_area_and_initialize;

VAR
        I                 : 1..Limit;

BEGIN

{ create the area, get a pointer to it and give it a name }

CREATE_AREA (   Area_identity,
                Start_of_area,
                'Work_space',
                STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_AREA' );

        FOR I := 1 TO Limit DO
            Start_of_area^ [I] := I;
END;

{--------------- END PROCEDURE DECLARATIONS --------------- }
```

```
PROGRAM Create_area_01 (OUTPUT);

BEGIN
        Issue_master_process_greeting;
        Create_area_and_initialize;
        Create_process_and_wait;

        Issue_master_process_greeting;
        Write_contents_of_area;
END.

{--------------- PROCESS BLOCK ---------------}

PROCESS_BLOCK Area_accessor;

VAR
        I                 : 1..Limit;

BEGIN
    WRITELN ( Process_greeting );

    Write_contents_of_area;

    FOR I := 1 TO Limit DO
        Start_of_area^ [I] := I * 5;

    WRITELN ( 'Process completed write and modification to area' );

END { PROCESS };
END { of MODULE };
```

E.5.1 Running COMM_5

Use the command procedure of the same name to build a system with COMM_5:

```
$        ! COMM_5.COM
$        ! Command procedure to build the VAXELN module COMM_5
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ Compile:
$        EPASCAL -
                 /LIST -
                 /DEBUG COMM_5, VAXELN-MODULES /LIBRARY
$ Link:
$        LINK -
                 /DEBUG COMM_5 -
                 ,VAXELN-MODULES  /LIBRARY -
                 ,ELN$:RTLSHARE /LIBRARY -
                 ,ELN$:RTL /LIBRARY /INCLUDE= -
                 (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$        SET NOVERIFY
$ System_build:
$        @ELN_EBUILD_1 COMM_5

$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program COMM_5 /debug
```

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
```

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

```
{
SOURCE:         COMM_6.PAS

PURPOSE:        Sets the system time then creates a job passing
                the current time to that job.

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_6

LINK:           $ LINK /DEBUG COMM_6, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_6

NOTES:          1) Command procedure COMM_6.COM compiles, links and
                   builds these programs into a system
______________________________________________________________________
}
MODULE Set_system_time [IDENT ('V1.000')];

VAR
        From_users_input_time           : VARYING_STRING (23);
        Status_from_call_to_SET_TIME    : INTEGER;
        The_current_binary_time_is      : LARGE_INTEGER;

{ -------------------------------------------------------------- }

PROCEDURE Prompt_user_for_time;

  BEGIN
        WRITELN ( 'Please enter time e.g. ',
                  '12-Dec-1986 15:47:30.00 > ' );
        READLN  ( From_users_input_time );
  END;

{ -------------------------------------------------------------- }

PROCEDURE And_get_the_binary_time;

 BEGIN
   The_current_binary_time_is :=
                TIME_VALUE ( From_users_input_time );
 END;

```

```
{ ------------------------------------------------------------- }

PROCEDURE Set_the_system_time;

 BEGIN
   SET_TIME     ( The_current_binary_time_is,
                  STATUS := Status_from_call_to_SET_TIME );

   IF NOT ODD   ( Status_from_call_to_SET_TIME ) THEN
        WRITELN (  'Call to SET_TIME failed with status: ',
                   Status_from_call_to_SET_TIME );
 END;

{ ------------------------------------------------------------- }

PROGRAM Set_system_time ( INPUT, OUTPUT );

VAR
        Time_for_new_job : VARYING_STRING(23);
        Status_returned  : INTEGER;
        Jobs_port        : PORT;

BEGIN
        Prompt_user_for_time;
        And_get_the_binary_time;
        Set_the_system_time;

{ get time string for argument to CREATE_JOB }

        And_get_the_binary_time;
        Time_for_new_job := TIME_STRING ( The_current_binary_time_is );

CREATE_JOB  ( Jobs_port,
              'COMM_6A',
              '',        { null arguments for number 1 & 2 }
              '',
              Time_for_new_job,
              STATUS := Status_returned );
           IF NOT ODD ( Status_returned ) THEN
              WRITELN ( 'Call to CREATE_JOB failed with status: ',
                        Status_returned );

END { of PROGRAM }.
END { of MODULE  };
```

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:

```
{
SOURCE:         COMM_6A.PAS

PURPOSE:        Gets the time from program argument and displays it plus
                30 days.

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_6A

LINK:           $ LINK /DEBUG COMM_6A, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT COMM_6

NOTES:          1) Command procedure COMM_6.COM compiles, links and
                   builds these programs into a system
______________________________________________________________________
}
MODULE Comm_6a [IDENT ('V1.000')];

CONST
        Argument_required = 3;
        Thirty_days       = '0030 00:00:00.00';

VAR
        Argument_3,
        Next_month_string : VARYING_STRING(23);
        Interval_30_days,
        Thirty_days_hence : LARGE_INTEGER;

{ ------------------------------------------------------------------ }

PROGRAM Comm_6a ( OUTPUT );

BEGIN

Argument_3 := PROGRAM_ARGUMENT ( Argument_required );

Interval_30_days  := TIME_VALUE ( Thirty_days );
```

```

{ SUBTRACT interval because delta times are held internally as
  NEGATIVE quadwords }
Thirty_days_hence := TIME_VALUE ( Argument_3 ) - Interval_30_days;

Next_month_string := TIME_STRING ( Thirty_days_hence );

WRITELN ( 'Program COMM_6A reporting...... ' );
WRITELN ( 'Time string passed to me was: ', Argument_3 );
WRITELN ( 'Adding 30 days to that gives: ', Next_month_string );

END { of PROGRAM }.
END { of MODULE  };
```

E.6.1 Running COMM_6

Use the command procedure of the same name to build a system with COMM_6:

```
$         ! COMM_6.COM
$         ! Command procedure to build the VAXELN modules COMM_6
$         ! and COMM_6A
$         !
$         ON ERROR THEN GOTO Switch_off_verify
$         SET DEFAULT Default_directory
$ !
$ Compile:
$                @ELN_COMPILE_1 COMM_6
$                @ELN_COMPILE_1 COMM_6A
$ Link:
$                @ELN_LINK_1    COMM_6
$                @ELN_LINK_1    COMM_6A
$ System_build:
$                @ELN_EBUILD_1   COMM_6
$ Switch_off_verify:
$         SET NOVERIFY
$         EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nonetwork /nofile /noserver
program COMM_6 /debug
program COMM_6A /norun /debug
```

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

```
{-----------------------------------------------------------------------
SOURCE:          COMM_7.PAS

PURPOSE:         To demonstrate creation of a circuit between VAXELN jobs

COMPILE:         $ EPASCAL /LIST /DEBUG COMM_7, -
                 _$  VAXELN-MODULES /LIBRARY

LINK:            $ LINK /DEBUG COMM_7, VAXELN-MODULES /LIBRARY, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY /INCLUDE= -
                 _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:           $ EBUILD /NOEDIT COMM_7

NOTES:           1) Command procedure COMM_7.COM compiles, links
                    and builds this module into a VAXELN system
----------------------------------------------------------------------}
MODULE Comm_7 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

TYPE
        Message_type = ^VARYING_STRING (50);

{------------------- PROGRAM BLOCK --------------------}

PROGRAM Comm_7 (OUTPUT);

VAR
        My_new_port,
        New_jobs_port       : PORT;
        Incoming_message    : Message_type;
        Message_identity    : MESSAGE;
        My_new_port_name    : NAME;
        Returned_status     : INTEGER;

```

```
{ MAIN program start }

BEGIN

{ could have used the default job port instead of creating one }

CREATE_PORT (   My_new_port,
                STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

{ name the port and put the name in the UNIVERSAL table }

CREATE_NAME (   My_new_port_name,
                'MASTER_PORT',
                My_new_port,
                TABLE  := NAME$UNIVERSAL
                STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_NAME' );

WRITELN ( 'First job about to accept circuit from second........' );

{ let the other job go }

INITIALIZATION_DONE ( STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'INITIALIZATION_DONE' );

{ process goes into wait state here for the connect circuit request
  to arrive...}

ACCEPT_CIRCUIT ( My_new_port,
                 STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'ACCEPT_CIRCUIT' );
```

```

{ wait for the message to arrive }

WAIT_ANY ( My_new_port,
           STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

WRITELN  ( '.......back in the master process about to read message ' );

{ collect the message and read it }

RECEIVE  (  Message_identity,
            Incoming_message,
            My_new_port,
            STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'RECEIVE' );

WRITELN ( 'Message: ', Incoming_message^ );

END { of PROGRAM }.
END { of MODULE };
```

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:

```
{---------------------------------------------------------------------
SOURCE:         COMM_7A.PAS

PURPOSE:        To demonstrate circuit connection with another job
                This program throws out a circuit to COMM_7

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_7A, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG COMM_7A, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT COMM_7

NOTES:          1) Command procedure COMM_7.COM compiles, links
                   and builds this module into a VAXELN system
--------------------------------------------------------------------}
MODULE Comm_7a [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

TYPE
        Message_type = ^VARYING_STRING (50);

{-------------------- PROGRAM BLOCK --------------------}

PROGRAM Comm_7a (OUTPUT);

VAR
        Returned_status         : INTEGER;
        Closedown               : Message_type;
        Message_identity        : MESSAGE;
        New_port                : PORT;

```

```
{ MAIN program start }

BEGIN

{ could use the job's port instead }

CREATE_PORT      (   New_port,
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

CONNECT_CIRCUIT (   New_port,
                     DESTINATION_NAME := 'MASTER_PORT',
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CONNECT_CIRCUIT' );

WRITELN ( 'This is from the new job' );

CREATE_MESSAGE  (   Message_identity,
                     Closedown,
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

Closedown^ := 'This is my closedown message';

SEND            (   Message_identity,
                     New_port,
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );
END { of PROGRAM }.
END { of MODULE };
```

E.7.1 Running COMM_7

Use the command procedure of the same name to build a system with COMM_7:

```
$        ! COMM_7.COM
$        ! Command procedure to build the VAXELN modules
$        ! COMM_7 and COMM_7A into a system
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ Compile1:
$        EPASCAL -
                /LIST -
                /DEBUG COMM_7, VAXELN-MODULES /LIBRARY
$ Compile2:
$        EPASCAL -
                /LIST -
                /DEBUG COMM_7A, VAXELN-MODULES /LIBRARY
$ Link1:
$        LINK -
                /DEBUG COMM_7 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link2:
$        LINK -
                /DEBUG COMM_7A -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$        EBUILD -
                /NOEDIT COMM_7
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
        CHARACTERISTIC /NOCONSOLE
        PROGRAM COMM_7 /INITIALIZE /DEBUG
        PROGRAM COMM_7A /DEBUG
```

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
```

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.

```
{-----------------------------------------------------------------------
SOURCE:         COMM_8.PAS

PURPOSE:        To demonstrate creation of a circuit between VAXELN
                jobs. This program 'talks' to COMM_8A. Each sends
                messages in a FOR loop, incrementing a message number
                count each time a message is prepared.

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_8, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG COMM_8, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT COMM_8

NOTES:          1) Command procedure COMM_8.COM compiles, links
                   and builds this module into a VAXELN system
---------------------------------------------------------------------}
MODULE Comm_8 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        LF    = ''(10);  { line-feed }
        Limit = 10;

TYPE
        Message_type = RECORD
           Message_number : 0..MAXINT;
           Data_array     : ARRAY[1..Limit] OF INTEGER;
        END;

```

```
{------------------- PROGRAM BLOCK -------------------}

PROGRAM Comm_8 (OUTPUT);

VAR
        My_new_port         : PORT;
        Outgoing_message,
        Incoming_message    : ^Message_type;
        Message_identity    : MESSAGE;
        My_new_port_name    : NAME;
        I,
        J,
        Returned_status     : INTEGER;

{ MAIN program start }

BEGIN

CREATE_PORT (    My_new_port,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

CREATE_NAME (    My_new_port_name,
                 'MASTER_PORT',
                 My_new_port,
                 TABLE  := NAME$UNIVERSAL,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_NAME' );

WRITELN ( 'This is from COMM_8...' );

INITIALIZATION_DONE ( STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'INITIALIZATION_DONE' );

{ wait for COMM_8A's circuit request to arrive...}

ACCEPT_CIRCUIT ( My_new_port,
                 STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'ACCEPT_CIRCUIT' );

```

```
FOR I := 1 TO 5 DO

BEGIN

CREATE_MESSAGE ( Message_identity,
                 Outgoing_message,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

WITH Outgoing_message^ DO

{ fill array with data and set message number }

BEGIN
   FOR J := 1 TO Limit DO Data_array[J] := J * I;
   IF I = 1 THEN Message_number := 1
            ELSE Message_number := Incoming_message^.Message_number + 1;
END;

SEND (  Message_identity,
        My_new_port,
        STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );

{ wait for a reply...}

WAIT_ANY ( My_new_port,
           STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

WRITELN  ( 'Back in COMM_8 about to read message...' );

RECEIVE  (  Message_identity,
            Incoming_message,
            My_new_port,
            STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'RECEIVE' );

```

```
WITH Incoming_message^ DO

{ output reply }

BEGIN
  WRITELN ( 'Message number: ', Message_number:1 );
  FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
  WRITELN ( LF );
END;

END; {FOR loop}

END { of PROGRAM }.
END { of MODULE };
```

This program, COMM_8A, throws out a circuit to COMM_8. It sends numbered messages to COMM_8.

```
{-----------------------------------------------------------------------
SOURCE:         COMM_8A.PAS

PURPOSE:        To demonstrate circuit connection with another job
                This program 'talks' to COMM_8 and runs after COMM_8
                calls INITIALIZATION_DONE.

COMPILE:        $ EPASCAL /LIST /DEBUG COMM_8A, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG COMM_8A, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT COMM_8

NOTES:          1) Command procedure COMM_8.COM compiles, links
                   and builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Create_circuit_a [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        Padding = '                                    ';
        LF      = ''(10);   { line-feed }
        Limit   = 10;

TYPE
        Message_type = RECORD
           Message_number : 0..MAXINT;
           Data_array     : ARRAY[1..Limit] OF INTEGER;
        END;

```

```
{-------------------- PROGRAM BLOCK --------------------}
PROGRAM Create_job_01 (OUTPUT);
VAR
        New_port            : PORT;
        Outgoing_message,
        Incoming_message    : ^Message_type;
        Message_identity    : MESSAGE;
        My_new_port_name    : NAME;
        I,
        J,
        Returned_status     : INTEGER;
{ MAIN program start }
BEGIN
CREATE_PORT     (   New_port,
                    STATUS := Returned_status );
            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );
CONNECT_CIRCUIT (   New_port,
                    DESTINATION_NAME := 'MASTER_PORT',
                    STATUS := Returned_status );
            Check_status_and_report ( Returned_status,
                                      'CONNECT_CIRCUIT' );
FOR I := 1 TO 5 DO
BEGIN
WAIT_ANY (New_port,
          STATUS := Returned_status );
            Check_status_and_report ( Returned_status,
                                      'WAIT_ANY' );
RECEIVE ( Message_identity,
          Incoming_message,
          New_port,
          STATUS := Returned_status );
            Check_status_and_report ( Returned_status,
                                        'RECEIVE' );
```

```
WRITELN ( Padding, 'This is from COMM_8A...' );

WITH Incoming_message^ DO
BEGIN
  WRITELN ( Padding, 'Message number is: ', Message_number:1 );
  WRITE   ( Padding );
  FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
  WRITELN ( LF );
END;

CREATE_MESSAGE  (   Message_identity,
                    Outgoing_message,
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

WITH Outgoing_message^ DO

{ multiply incoming data for reply and set new message number }

BEGIN
  Message_number := Incoming_message^.Message_number + 1;
  FOR J := 1 TO Limit DO Data_array[J] :=
                         Incoming_message^.Data_array[J] * 2;
END;

SEND            (   Message_identity,
                    New_port,
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );
END;

END { of PROGRAM }.
END { of MODULE };
```

E.8.1 Running COMM_8

Use the command procedure of the same name to build a system with COMM_8:

```
$       ! COMM_8.COM
$       ! Command procedure to build the VAXELN modules
$       ! COMM_8 and COMM_8A and build them into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile1:
$       EPASCAL -
                /LIST -
                /DEBUG COMM_8, VAXELN-MODULES /LIBRARY
$ Compile2:
$       EPASCAL -
                /LIST -
                /DEBUG COMM_8A, VAXELN-MODULES /LIBRARY
$ Link1:
$       LINK -
                /DEBUG COMM_8 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link2:
$       LINK -
                /DEBUG COMM_8A -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$       SET NOVERIFY
$ System_build:
$               @ELN_EBUILD_1 COMM_8
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
        CHARACTERISTIC /NOCONSOLE
        PROGRAM COMM_8 /INITIALIZE /DEBUG
        PROGRAM COMM_8A /DEBUG
```

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
```

```
!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
```

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.

```
{-------------------------------------------------------------------------
SOURCE:          COMM_9.PAS

PURPOSE:         To demonstrate creation of a circuit between VAXELN
                 jobs with an EXPEDITED message SEND

COMPILE:         $ EPASCAL /LIST /DEBUG COMM_9, -
                 _$  VAXELN-MODULES /LIBRARY

LINK:            $ LINK /DEBUG COMM_9, VAXELN-MODULES /LIBRARY, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY /INCLUDE= -
                 _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:           $ EBUILD /NOEDIT COMM_9

NOTES:           1) Command procedure COMM_9.COM compiles, links
                    and builds this module into a VAXELN system
------------------------------------------------------------------------}
MODULE Create_circuit [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        LF         = ''(10);     { line-feed }
        Limit      = 10;
        Port_limit = 8;

TYPE
        Message_type = RECORD
           Message_number : 0..MAXINT;
           Data_array     : ARRAY[1..Limit] OF INTEGER;
        END;

        Expedited_type = RECORD
           Expedite_string : VARYING_STRING(16);
        END;

```

```
{------------------- PROGRAM BLOCK -------------------}

PROGRAM Create_job_01 (OUTPUT);

VAR
        My_new_port           : PORT;
        Expedited_message     : ^Expedited_type;
        Outgoing_message      : ^Message_type;
        Message_identity      : MESSAGE;
        My_new_port_name      : NAME;
        I,
        J,
        Returned_status       : INTEGER;

{ MAIN program start }

BEGIN

CREATE_PORT (    My_new_port,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

CREATE_NAME (    My_new_port_name,
                 'MASTER_PORT',
                 My_new_port,
                 TABLE  := NAME$UNIVERSAL,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_NAME' );

WRITELN ( 'This is from COMM_9...' );

INITIALIZATION_DONE ( STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'INITIALIZATION_DONE' );

ACCEPT_CIRCUIT ( My_new_port,
                 STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'ACCEPT_CIRCUIT' );
```

```
{ start pushing messages along the circuit }

FOR I := 1 TO Port_limit DO

BEGIN

CREATE_MESSAGE ( Message_identity,
                 Outgoing_message,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );


WITH Outgoing_message^ DO

  BEGIN
    FOR J := 1 TO Limit DO Data_array[J] := J * I;
    Message_number := I;
  END;

SEND (  Message_identity,
        My_new_port,
        STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );
END;

{ having reached the limit for the port issue an EXPEDITED message }

CREATE_MESSAGE ( Message_identity,
                 Expedited_message,
                 STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

```

```
WITH Expedited_message^ DO
        Expedite_string := 'EXPEDITED !!!!!!';

{ the value 16 in the SIZE argument is required otherwise
  the previous message size is assumed. A value > 16 will be
  an error for a message typed EXPEDITED }

SEND (  Message_identity,
        My_new_port,
        SIZE     := 16,
        EXPEDITE := TRUE,
        STATUS   := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );

END { of PROGRAM }.
END { of MODULE };
```

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.

```
{---------------------------------------------------------------------
SOURCE:          COMM_9A.PAS

PURPOSE:         To demonstrate circuit connection with another job and
                 handling of an expedited message

COMPILE:         $ EPASCAL /LIST /DEBUG COMM_9A, -
                 _$  VAXELN-MODULES /LIBRARY

LINK:            $ LINK /DEBUG COMM_9A, VAXELN-MODULES /LIBRARY, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY /INCLUDE= -
                 _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:           $ EBUILD /NOEDIT COMM_9

NOTES:           1) Command procedure COMM_9.COM compiles, links
                    and builds this module into a VAXELN system
--------------------------------------------------------------------}
MODULE Create_circuit_a [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        Padding    = '                                        ';
        Port_limit = 8;
        LF         = ''(10);   { line-feed }
        Limit      = 10;
        Interval   = '0000 00:00:10.00';

TYPE
        Message_type = RECORD
           Message_number : INTEGER;
           Data_array     : ARRAY[1..Limit] OF INTEGER;
        END;

VAR
        KER$_EXPEDITED,
        KER$_DISCONNECT,
        KER$_SUCCESS      : [EXTERNAL, VALUE] INTEGER;
```

```
{-------------------- PROGRAM BLOCK --------------------}

PROGRAM Create_job_01 (OUTPUT);

VAR
        Binary_interval     : LARGE_INTEGER;
        Expedited_message   : VARYING_STRING(44);
                              { 44 is length of MESSAGE_TYPE }
        New_port            : PORT;
        Incoming_message    : ^Message_type;
        Message_identity    : MESSAGE;
        My_new_port_name    : NAME;
        I,
        J,
        Returned_status     : INTEGER;

{ MAIN program start }

BEGIN

CREATE_PORT     (   New_port,
                    LIMIT  := Port_limit,
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

CONNECT_CIRCUIT (   New_port,
                    DESTINATION_NAME := 'MASTER_PORT',
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CONNECT_CIRCUIT' );

{ place process in wait state so that PORT will become filled
  with unread messages from COMM_9 }

Binary_interval := TIME_VALUE ( Interval );

WAIT_ANY (TIME := Binary_interval,
          STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'WAIT_ANY' );

```

```
FOR I := 1 TO 10 DO
BEGIN

WRITELN ( Padding, 'This is from COMM_9A...' );

RECEIVE ( Message_identity,
          Incoming_message,
          New_port,
          STATUS := Returned_status );

  { Usual status checks are replaced by specific checks...
    Can't use the KER$ codes with CASE statement - won't compile with
    this TYPE for a CASE because they are not of the same type as the
    CASE expression }

IF Returned_status = KER$_EXPEDITED THEN
  BEGIN
     WRITELN ( Padding, '***** EXPEDITED MESSAGE RECEIVED *****' );

   { typecast to 42 bytes because varying_string has 16 bit header too -
     the message type has a total of 44 bytes allocated to it.
     Must be typecast because the data type of the message coming in
     is incompatible with the buffer in this program }

     Expedited_message := Incoming_message^::VARYING_STRING(42);
     WRITELN ( Padding, 'Expedited message is: ' );
     WRITELN ( Padding, Expedited_message, LF );
  END;

IF Returned_status = KER$_SUCCESS THEN

  WITH Incoming_message^ DO
  BEGIN
    WRITELN ( Padding, 'Message number is: ', Message_number:1 );
    WRITE   ( Padding );
    FOR J := 1 TO Limit DO WRITE ( Data_array[J]:4 );
    WRITELN ( LF );
  END;

IF Returned_status = KER$_DISCONNECT THEN
  WRITELN ( Padding, 'Job COMM_9 has disconnected the circuit...' );

END;

END { of PROGRAM }.
END { of MODULE };
```

E.9.1 Running COMM_9

Use the command procedure of the same name to build a system with COMM_9:

```
$       ! COMM_9.COM
$       ! Command procedure to build the VAXELN modules
$       ! COMM_9 and COMM_9A and build them into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile1:
$       EPASCAL -
                /LIST -
                /DEBUG COMM_9, VAXELN-MODULES /LIBRARY
$ Compile2:
$       EPASCAL -
                /LIST -
                /DEBUG COMM_9A, VAXELN-MODULES /LIBRARY
$ Link1:
$       LINK -
                /DEBUG COMM_9 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link2:
$       LINK -
                /DEBUG COMM_9A -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$       SET NOVERIFY
$ System_build:
$               @ELN_EBUILD_1 COMM_9
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
        CHARACTERISTIC /NOCONSOLE
        PROGRAM COMM_9 /INITIALIZE /DEBUG
        PROGRAM COMM_9A /DEBUG
```

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
```

```
!                                      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.

```
{-----------------------------------------------------------------------
SOURCE:         NET_1.PAS

PURPOSE:        This program loads, dynamically, TODAY.EXE from the
                host VAX and runs the program in a new job created
                by this program for it
                TODAY.EXE, written in VAX FORTRAN returns:
                  a) day
                  b) month
                  c) year
                  d) week
                  e) year day
                  f) century day
                for the current date

COMPILE:        $ EPASCAL /LIST /DEBUG NET_1, VAXELN-MODULES /LIB -
                _$ ,ELN$:RTLOBJECT /LIBRARY

                $ FORTRAN /LIST /NOOPTIMIZE /DEBUG TODAY

```

```
LINK:           $ LINK /DEBUG NET_1, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

                $ LINK /DEBUG /NOSYSLIB TODAY, -
                _$ ELN$:FRTLOBJECT /LIBRARY, RTLSHARE /LIBRARY, -
                _$ RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT NET_1

NOTES:          1) Command procedure NET_1.COM compiles, links and
                   builds these programs into a system
----------------------------------------------------------------------
}
MODULE Set_system_time [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report,
        $LOADER_UTILITY;

CONST
        Wait_period       = '0000 00:00:05.00';
VAR
        Status_returned   : INTEGER;
        Jobs_port         : PORT;
        Program_name      : VARYING_STRING(40) := 'TODAY';

        From_users_input_time          : VARYING_STRING (23);
        Status_from_call_to_SET_TIME   : INTEGER;
        Binary_interval,
        The_current_binary_time_is     : LARGE_INTEGER;
        Loaded_jobs_port               : PORT;

{ ------------------------------------------------------------ }
```

```

PROCEDURE Prompt_user_for_time;

  BEGIN
        WRITELN ( 'Please enter time e.g. ',
                  '12-Dec-1986 15:47:30.00 > ' );
        READLN  ( From_users_input_time );
  END;

{ ------------------------------------------------------------- }

PROCEDURE And_get_the_binary_time;

 BEGIN
   The_current_binary_time_is := TIME_VALUE ( From_users_input_time );
 END;

{ ------------------------------------------------------------- }

PROCEDURE Set_the_system_time;

 BEGIN
   SET_TIME     ( The_current_binary_time_is,
                  STATUS := Status_from_call_to_SET_TIME );

   IF NOT ODD   ( Status_from_call_to_SET_TIME ) THEN
        WRITELN (  'Call to SET_TIME failed with status: ',
                   Status_from_call_to_SET_TIME );
 END;

{ ------------------------------------------------------------- }

PROGRAM Set_system_time ( INPUT, OUTPUT );

BEGIN
        Prompt_user_for_time;
        And_get_the_binary_time;
        Set_the_system_time;
```

```

ELN$LOAD_PROGRAM ( '1.241::DISK$COURSEDSK:[SHONE]TODAY.EXE',
                   Program_name,
                   FALSE,                  {Kernel_mode}
                   TRUE,                   {Start_with_debug}
                   FALSE,                  {Power_recovery}
                   2,                      {Kernel_stack_size}
                   1,                      {Initial_user_stack_size}
                   0,                      {Message_limit}
                   16,                     {Job_priority}
                   8,                      {Process_priority}
                   STATUS := Status_returned );

        Check_status_and_report ( Status_returned,
                                  'ELN$LOAD_PROGRAM' );

CREATE_JOB ( Loaded_jobs_port,
             Program_name,
             STATUS := Status_returned );

        Check_status_and_report ( Status_returned,
                                  'CREATE_JOB' );

Binary_interval := TIME_VALUE ( Wait_period );

WAIT_ANY   ( TIME   := Binary_interval,
            STATUS := Status_returned );

        Check_status_and_report ( Status_returned,
                                  'WAIT_ANY' );

END { of PROGRAM }.
END { of MODULE  };
```

F.1.1 Running NET_1

Use the command procedure of the same name to build a system with NET_1:

```
$       ! NET_1.COM
$       ! Command procedure to build the EPASCAL module NET_1
$       ! and the VAX FORTRAN program TODAY into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile_NET_1:
$       EPASCAL -
                /LIST -
                /DEBUG NET_1, VAXELN-MODULES /LIBRARY -
                ,ELN$:RTLOBJECT /LIBRARY
$ Compile_Today:
$       FORTRAN -
                /LIST -
                /NOOPTIMIZE -
                /DEBUG TODAY
$ Link_NET_1:
$       LINK -
                /DEBUG NET_1 -
                ,VAXELN-MODULES /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link_Today:
$       LINK -
                /DEBUG -
                /NOSYSLIB TODAY -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,ELN$:RTLSHARE /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT NET_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /image_list=(PRGLOADER)
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 FALSERVER, 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;
```

```
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$SCRATCH }
{                                                                              }
  INIT_VOLUME('DUA1',
             'SCRATCH',
              VERIFIED := FALSE,
              BAD_LIST := bad_block_list::DSK$_BADLIST(0),
              STATUS   := result);
  ELN$GET_STATUS_TEXT(result, which_flag, result_string);
  WRITELN('The status of initializing the volume was');
  WRITELN(result_string);
{                                                                              }
{ Mount the volume and get the status                                          }
{                                                                              }
  MOUNT_VOLUME('DUA1',,result);
  ELN$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);
```

```
{                                                                              }
{ 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 Running FILE_1

Use the command procedure of the same name to build a system with FILE_1

```
$        ! FILE_1.COM
$        ! Command procedure to build the VAXELN module FILE_1
$        ! into a system
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ Compile:
$        EPASCAL -
                 /LIST -
                 /DEBUG FILE_1, ELN$:RTLOBJECT /LIBRARY
$ Link:
$        LINK -
                 /DEBUG FILE_1 -
                 ,ELN$:RTLSHARE   /LIBRARY -
                 ,ELN$:RTL /LIBRARY /INCLUDE= -
                 (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$        EBUILD -
                         /NOEDIT FILE_1
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program FILE_1 /debug
device DUA /register=%O772150 /vector=%O154
```

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:

```
*-----------------------------------------------------------------------
* SOURCE:                FILE_2.FOR
*
* PURPOSE:         This VAX FORTRAN program shows remote file access
*                  from VAXELN. It reads a file called ELEMENTS.DAT
*                  whose contents look like:
*
*                        Ac 89227.00000Actinium
*                        Al 13 26.98150Aluminium
*                        Am 95243.00000Americium
*                          . . . . . . . . . .
*                          . . . . . . . . . .
*                        Yb 70173.04000Ytterbium
*                        Y  39 88.90500Yttrium
*                        Zn 30 65.37000Zinc
*                        Zr 40 91.22000Zirconium
*
*
* COMPILE:       $ FORTRAN /LIST /DEBUG /NOOPTIMIZE FILE_2
*
* LINK:          $ LINK /DEBUG FILE_2, -
*                _$ ELN$:FRTLOBJECT /LIBRARY, -
*                _$ ELN$:RTLSHARE /LIBRARY, -
*                _$ ELN$:RTL /LIBRARY
*
* BUILD:         $ EBUILD /NOEDIT FILE_2
*
* NOTES:         1) Command procedure FILE_2.COM compiles, links
*                   and builds this module into a VAXELN system
* ----------------------------------------------------------------------
```

```

      PROGRAM VAXELN_file_IO

      IMPLICIT NONE

      INTEGER         First_element,
      1               Last_element,
      1               Unit_number,
      1               Terminal

* When 'they' invent more elements change this parameter value

      PARAMETER       ( First_element  =   1,
      1                 Last_element   = 103,
      1                 Unit_number    =   4,
      1                 Terminal       =   6 )

* change this file spec if not on node 1.241 and device and
* directory details are different

      CHARACTER *50 File_spec
      1               /'1.241::DISK$COURSEDSK:[SHONE]ELEMENTS.DAT'/

      CHARACTER       Symbol *2,
      1               Element_name *12

      INTEGER         Element_number,
      1               Atomic_number

      REAL            Atomic_weight

      OPEN    ( Unit_number,
      1         FILE     = File_spec,
      1         STATUS   = 'OLD' )

* Read file and display to terminal

      DO Element_number = First_element, Last_element, 1

        READ ( Unit_number,
      1        '(A, I3, F9.5, A)',
      1        END = 999 )
      1       Symbol, Atomic_number, Atomic_weight, Element_name

* subroutine call to convert lowercase to uppercase
* in 'Element_name'

        CALL LC2UC (  Element_name,
      1               Element_name )

```

```
               WRITE ( Terminal,
          1          '(1X, ''Details of element: '', A /
          1            1X, ''Symbol: '', A,
          1                '',  atomic number: '', I3,
          1                '',  atomic weight: '', F9.5 / )')
          1        Element_name, Symbol, Atomic_number, Atomic_weight

          END DO

    999   CLOSE ( Unit_number )
          END

          SUBROUTINE LC2UC (INSTR,OUTSTR)
*
*         Subroutine to convert lowercase letters to uppercase
*
*         Input required is string and output location
*
          IMPLICIT NONE

          INTEGER          K,
          1                Key
          CHARACTER        INSTR*(*),
          1                OUTSTR*(*),
          1                Lcase *26,
          1                Ucase *26

          DATA Lcase /'abcdefghijklmnopqrstuvwxyz'/
          DATA Ucase /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/

          DO K = 1, LEN(INSTR), 1

            Key = INDEX (Lcase, Instr(K:K))

            IF (Key .GT. 0) THEN
               Outstr(K:K) = Ucase(Key:Key)
            ELSE
               Outstr(K:K) = Instr(K:K)
            END IF

          END DO

          RETURN
          END
```

G.2.1 Running FILE_2

Use the command procedure of the same name to build a system with FILE_2

```
$       ! FILE_2.COM
$       ! Command procedure to build the VAXELN module FILE_2
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       FORTRAN -
                /LIST -
                /NOOPTIMIZE -
                /DEBUG FILE_2
$ Link:
$       LINK -
                /DEBUG FILE_2 -
                ,ELN$:FRTLOBJECT /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY
$ System_build:
$       EBUILD -
                        /NOEDIT FILE_2
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program FILE_2 /debug
```

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.
```

```
{
{ 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

    lpv11             : ^lpv11_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 );
```

```
{ 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         := lpv11,
                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
```

```
{
{  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;


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 );
```

```
            { 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 ( lpv11^.dbr,
                                 data := char_out );
                csr := read_register ( lpv11^.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

    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
```

```
{               buffer_length           - length of data in buffer
{               context                 - driver specific parameter (unused)
{
{ Outputs:
{
{       return value - write status
{--}

var
    write_status : boolean;

begin

    write_register ( lpv11^.csr, inte := true );
    write_status := write_chars ( lp_structure,
                                  buffer,
                                  buffer_length,
                                  saved_record_attributes,
                                  put_chars );
    write_register ( lpv11^.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;
```

```
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('DUA1:[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$GET_STATUS_TEXT(result, which_flag, result_text);
  WRITELN(result_text);
  WRITELN;
  NEW(pointer_to_directory);
{                                                                        }
{ Write text into a file.                                                }
{                                                                        }
  OPEN(working_file,
       FILE_NAME     := 'DUA1:[TEMP]TEST.TXT',
       HISTORY       := HISTORY$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     := 'DUA1:[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;
```

```
  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, 'DUA1:[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.
```

I.1.1 Running EXCEPT_1

Use the command procedure of the same name to build a system with EXCEPT_1

```
$       ! EXCEPT_1.COM
$       ! Command procedure to build the VAXELN module EXCEPT_1
$       ! into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG EXCEPT_1, ELN$:RTLOBJECT /LIBRARY
$ Link:
$       LINK -
                /DEBUG EXCEPT_1 -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT EXCEPT_1
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /volumes=("DUA1 SCRATCH")
program EXCEPT_1 /debug
device DUA /register=%O772150 /vector=%O154
```

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

```
{-----------------------------------------------------------------------
SOURCE:          LAB_1.PAS

PURPOSE:         To calculate temperatures on one of four scales:
                   1) Celsius      (ice melts: 0.0, water boils 100.0)
                   2) Fahrenheit   ( 32.0, 212.0)
                   3) Kelvin       (273.0, 373.0)
                   4) Reaumur      (  0.0,  80.0)
                 given an input on one of these scales.

COMPILE:         $ EPASCAL /LIST /DEBUG LAB_1

LINK:            $ LINK /DEBUG LAB_1, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT LAB_1

NOTES:           1) Command procedure LAB_1.COM compiles, links
                    and builds this module into a VAXELN system or
                    for running under VMS
---------------------------------------------------------------------}
MODULE Lab_1 [IDENT ('V1.000')];

CONST
        LF      = ''(10);   {line-feed}

VAR
        User_continues  : BOOLEAN := TRUE;
        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Answer,
        Temp_scale_in,
        Temp_scale_out  : CHAR;
```

```

{--------------- FUNCTION DECLARATIONS --------------- }

FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;

FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
    END;

{----------------------------------------------------- }

FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Kelvin_to_Celsius := T - 273.0;
    END;

{----------------------------------------------------- }

FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Reaumur_to_Celsius := T * 1.25;
    END;

{----------------------------------------------------- }

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Fahrenheit := T * 1.8 + 32.0;
    END;

{----------------------------------------------------- }

FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Kelvin := T + 273.0;
    END;

{----------------------------------------------------- }

FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Reaumur := T * 0.8;
    END;

{------------- END FUNCTION DECLARATIONS ------------- }
```

```
{------------------ PROGRAM BLOCK -------------------}
PROGRAM Lab_1 (OUTPUT);
   PROCEDURE Ensure_uppercase ( VAR C : CHAR );
   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};
BEGIN
WHILE User_continues DO
BEGIN
  WRITELN ( LF );
  WRITE  ( 'Please enter temperature: ' );
  READLN ( Temp_value_in );
  WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
  READLN ( Temp_scale_in );
  Ensure_uppercase ( Temp_scale_in );
  WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
  READLN ( Temp_scale_out );
  Ensure_uppercase ( Temp_scale_out );
{ convert everything to Celsius (or leave it in Celsius) }
CASE Temp_scale_in OF
        'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
        'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );
        'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );
        'C' : Temp := Temp_value_in;
END;  {CASE Temp_scale_in}
{ provide conversion }
CASE Temp_scale_out OF
        'C'  : Temp_value_out := Temp;
        'F'  : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
        'R'  : Temp_value_out := Celsius_to_Reaumur    ( Temp );
        'K'  : Temp_value_out := Celsius_to_Kelvin     ( Temp );
END; {CASE temp_scale_out}
```

```

WRITELN (   'Temperature before conversion: ',
            Temp_value_in :6:1,
            ' deg',
            Temp_scale_in );

WRITELN (   'Temperature after conversion:  ',
            Temp_value_out :6:1,
            ' deg',
            Temp_scale_out );

WRITE  ( 'Do you wish to try again? [Y/N]: ' );
READLN (  Answer );

User_continues := Answer IN ['Y', 'y'];

END;

END {of PROGRAM}.
END {of MODULE };
```

K.1.1 Running LAB_1

```
$        ! LAB_1.COM
$        ! Command procedure to build the VAXELN
$        ! module LAB_1
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$        IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$        SET VERIFY
$          EPASCAL /LIST LAB_1
$          LINK LAB_1
$        GOTO Switch_off_verify
$ !
$ Compile:
$               @ELN_COMPILE_1  LAB_1
$ Link:
$               @ELN_LINK_1     LAB_1
$ System_build:
$               @ELN_EBUILD_1   LAB_1
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program LAB_1 /debug
```

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

```
* SOURCE:               LAB_1_F.FOR
*
* PURPOSE:        To calculate temperatures on one of four scales:
*                 1) Celsius      (ice melts: 0.0, water boils 100.0)
*                 2) Fahrenheit   ( 32.0, 212.0)
*                 3) Kelvin       (273.0, 373.0)
*                 4) Reaumur      (  0.0,  80.0)
*               given an input on one of these scales.
*
* COMPILE:      $ FORTRAN /LIST /DEBUG /NOOPTIMIZE LAB_1_F
*
* LINK:         $ LINK /DEBUG /NOSYSLIB LAB_1_F, -
*               _$ ELN$:FRTLOBJECT /LIBRARY, -
*               _$ ELN$:RTLSHARE /LIBRARY, -
*               _$ ELN$:RTL /LIBRARY
*
* BUILD:        $ EBUILD /NOEDIT LAB_1_F
*
* NOTES:        1) Command procedure LAB_1_F.COM compiles, links
*               and builds this module into a VAXELN system or
*               for running under VMS
*

        IMPLICIT NONE

        LOGICAL         User_continues /.TRUE./

        REAL            Temp,
        1               Temp_value_in,
        1               Temp_value_out,
        1               Fahrenheit_to_Celsius,
        1               Reaumur_to_Celsius,
        1               Kelvin_to_Celsius,
        1               Celsius_to_Fahrenheit,
        1               Celsius_to_Kelvin,
        1               Celsius_to_Reaumur

        CHARACTER       Answer,
        1               Temp_scale_in,
        1               Temp_scale_out

```

```
      DO WHILE (User_continues)

      WRITE ( 6, '(/1X, ''Please enter temperature: '' $)')
      READ  ( 5, * ) Temp_value_in

      WRITE ( 6, '(1X, ''Please enter scale for that '',
      1                ''temperature [K,C,F,R]: '' $)')
      READ  ( 5, '(A)' ) Temp_scale_in

* convert scale to uppercase if necessary
      CALL LC2UC ( Temp_scale_in, Temp_scale_in )

      WRITE ( 6, '(1X, ''Please enter the conversion '',
      1                ''required [K,C,F,R]: '' $)')
      READ  ( 5, '(A)' ) Temp_scale_out

* convert scale to uppercase if necessary
      CALL LC2UC ( Temp_scale_out, Temp_scale_out )

* convert everything to Celsius or leave it in Celsius

      IF (Temp_scale_in .EQ. 'F')
      1       Temp = Fahrenheit_to_Celsius ( Temp_value_in )

      IF (Temp_scale_in .EQ. 'R')
      1       Temp = Reaumur_to_Celsius ( Temp_value_in )

      IF (Temp_scale_in .EQ. 'K')
      1       Temp = Kelvin_to_Celsius ( Temp_value_in )

      IF (Temp_scale_in .EQ. 'C')
      1       Temp = Temp_value_in


* provide conversion *

      IF (Temp_scale_out .EQ. 'C')
      1           Temp_value_out = Temp

      IF (Temp_scale_out .EQ. 'F')
      1           Temp_value_out = Celsius_to_Fahrenheit ( Temp )

      IF (Temp_scale_out .EQ. 'R')
      1           Temp_value_out = Celsius_to_Reaumur ( Temp )

      IF (Temp_scale_out .EQ. 'K')
      1           Temp_value_out = Celsius_to_Kelvin ( Temp )
```

```

      WRITE ( 6, '(1X, ''Temperature before conversion: '',
     1                 F6.1, '' deg'', A )')
     1        Temp_value_in, Temp_scale_in

      WRITE ( 6, '(1X, ''Temperature after conversion:  '',
     1                 F6.1, '' deg'', A )')
     1        Temp_value_out, Temp_scale_out

      WRITE  ( 6, '(1X, ''Do you wish to try again? [Y/N]: '' $)')
      READ   ( 5, '(A)') Answer

      User_continues = Answer .EQ. 'Y' .OR. Answer .EQ. 'y'

      END DO

      END

* -----------------------------------------------------

      REAL FUNCTION Fahrenheit_to_Celsius (T)

      IMPLICIT NONE
      REAL    T

      Fahrenheit_to_Celsius = (T - 32.0) * 5.0 / 9.0

      RETURN
      END

* -----------------------------------------------------

      REAL FUNCTION Kelvin_to_Celsius (T)

      IMPLICIT NONE
      REAL    T

      Kelvin_to_Celsius = T - 273.0

      RETURN
      END

```

```
* --------------------------------------------------------

      REAL FUNCTION Reaumur_to_Celsius (T)

      IMPLICIT NONE
      REAL    T

      Reaumur_to_Celsius = T * 1.25

      RETURN
      END

* --------------------------------------------------------

      REAL FUNCTION Celsius_to_Fahrenheit (T)

      IMPLICIT NONE
      REAL    T

      Celsius_to_Fahrenheit = T * 1.8 + 32.0

      RETURN
      END

* --------------------------------------------------------

      REAL FUNCTION Celsius_to_Kelvin (T)

      IMPLICIT NONE
      REAL    T

      Celsius_to_Kelvin = T + 273.0

      RETURN
      END

* --------------------------------------------------------

      REAL FUNCTION Celsius_to_Reaumur (T)

      IMPLICIT NONE
      REAL    T

      Celsius_to_Reaumur = T * 0.8

      RETURN
      END
```

```
* ----------------------------------------------------

        SUBROUTINE LC2UC (INSTR,OUTSTR)
*
*       Subroutine to convert lowercase letters to uppercase
*
*       Input required is string and output location
*
        INTEGER         K,
        1               Key
        CHARACTER       INSTR*(*),
        1               OUTSTR*(*),
        1               Lcase *26,
        1               Ucase *26

        DATA Lcase /'abcdefghijklmnopqrstuvwxyz'/
        DATA Ucase /'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/

        DO K = 1, LEN(INSTR), 1

          Key = INDEX (Lcase, Instr(K:K))

          IF (Key .GT. 0) THEN
             Outstr(K:K) = Ucase(Key:Key)
          ELSE
             Outstr(K:K) = Instr(K:K)
          END IF

        END DO

        RETURN
        END
```

K.2.1 Running LAB_1_F.FOR

Use the command procedure of the same name to build a system with LAB_1_F:

```
$        ! LAB_1_F.COM
$        ! Command procedure to compile and link the VAXELN
$        ! module LAB_1_F
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        INQUIRE -
                 /NOPUNCTUATION  Running_under_VMS -
                 "Do you wish to run this program under VMS [Y/N]: "
$        IF .NOT. Running_under_VMS THEN GOTO Compile
$ !
$        SET VERIFY
$          FORTRAN /LIST LAB_1_F
$          LINK LAB_1_F
$        GOTO Switch_off_verify
$ !
$ Compile:
$                FORTRAN -
                         /LIST -
                         /DEBUG -
                         /NOOPTIMIZE LAB_1_F
$ Link:
$                LINK -
                         /NOSYSLIB -
                         /DEBUG LAB_1_F -
                         ,ELN$:FRTLOBJECT /LIBRARY -
                         ,RTLSHARE /LIBRARY -
                         ,RTL /LIBRARY
$        SET NOVERIFY
$ System_build:
$                @ELN_EBUILD_1   LAB_1_F
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
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 FALSERVER, 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
```

K.3 SOLUTION TO EXERCISE 1 - LAB_1_C.C

```
/* ---------------------------------------------------------------------
 * SOURCE:              LAB_1_C.C
 *
 * PURPOSE:         To calculate temperatures on one of four scales:
 *                   1) Celsius      (ice melts: 0.0, water boils 100.0)
 *                   2) Fahrenheit   ( 32.0, 212.0)
 *                   3) Kelvin       (273.0, 373.0)
 *                   4) Reaumur      (  0.0,  80.0)
 *                   given an input on one of these scales.
 *
 * COMPILE:       $ CC /LIST /DEBUG LAB_1_C + ELN$:VAXELNC /LIBRARY
 *
 * LINK:          $ LINK /DEBUG LAB_1_C, -
 *               _$ ELN$:CRTLSHARE /LIBRARY, -
 *               _$ ELN$:RTLSHARE /LIBRARY, -
 *               _$ ELN$:RTL /LIBRARY
 *
 * BUILD:         $ EBUILD /NOEDIT LAB_1_C
 *
 * NOTES:       1) Command procedure LAB_1_C.COM compiles, links
 *                 and builds this module into a VAXELN system or
 *                 for running under VMS
 * --------------------------------------------------------------------
 */

#include $vaxelnc
#include ctype          /* for the toupper function */

BOOLEAN user_continues = TRUE;

float   temp,
        temp_value_in,
        temp_value_out;

char    answer,
        temp_scale_in,
        temp_scale_out;

```

```
lab_1 ()
        MAIN_PROGRAM
{
float   fahrenheit_to_celsius(),
        kelvin_to_celsius(),
        reaumur_to_celsius(),
        celsius_to_fahrenheit(),
        celsius_to_kelvin(),
        celsius_to_reaumur();

char    dummy;

while (user_continues)
    {
    printf ("\n\n");
    printf ("Please enter temperature: ");
    scanf  ("%f", &temp_value_in);

    dummy = getchar();     /* dispose of end-of-line */
    printf ("\nPlease enter scale for that temperature [K,C,F,R]: ");
    scanf  ("%c",  &temp_scale_in);

/*      uppercase temp_scale_in as necessary
 */
    temp_scale_in = toupper (temp_scale_in);

    dummy = getchar();     /* dispose of end-of-line */
    printf ("Please enter the conversion required [K,C,F,R]: ");
    scanf  ("%c",  &temp_scale_out);

/*      uppercase temp_scale_out as necessary
 */
    temp_scale_out = toupper (temp_scale_out);

    dummy = getchar();     /* dispose of end-of-line */

switch (temp_scale_in )
        {
        case 'F': temp = fahrenheit_to_celsius (temp_value_in);
                  break;
        case 'R': temp = reaumur_to_celsius    (temp_value_in);
                  break;
        case 'K': temp = kelvin_to_celsius     (temp_value_in);
                  break;
        case 'C': temp = temp_value_in;
                  break;
        } /* switch temp_scale_in */

```

```
switch (temp_scale_out)
        {
        case 'C': temp_value_out = temp;
                  break;
        case 'F': temp_value_out = celsius_to_fahrenheit (temp);
                  break;
        case 'R': temp_value_out = celsius_to_reaumur (temp);
                  break;
        case 'K': temp_value_out = celsius_to_kelvin (temp);
                  break;
        } /* switch temp_scale_out */

printf ("Temperature before conversion: %6.1f deg%c\n",
        temp_value_in, temp_scale_in);
printf ("Temperature after conversion:  %6.1f deg%c\n",
        temp_value_out, temp_scale_out);

printf ("Do you wish to try again? [Y/N]: ");
scanf  ("%c", &answer);

        user_continues = (answer == 'Y' || answer == 'y');

        } /* while user continues */

}

/*    ========== Temperature conversion functions ==========
 */

float fahrenheit_to_celsius (t)
float t;
        {
        return ((t-32.0) * 5.0 / 9.0);
        }
```

```
/* -------------------------------------------------- */

float kelvin_to_celsius (t)
float t;
        {
        return (t - 273.0);
        }

/* -------------------------------------------------- */

float reaumur_to_celsius (t)
float t;
        {
        return (t * 1.25);
        }

/* -------------------------------------------------- */

float celsius_to_fahrenheit (t)
float t;
        {
        return (t * 1.8 + 32.0);
        }

/* -------------------------------------------------- */

float celsius_to_kelvin (t)
float t;
        {
        return (t + 273.0);
        }

/* -------------------------------------------------- */

float celsius_to_reaumur (t)
float t;
        {
        return (t * 0.8);
        }
```

K.3.1 Running LAB_1_C

```
$       ! LAB_1_C.COM
$       ! Command procedure to build the VAXELN module LAB_1_C
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       INQUIRE -
                /NOPUNCTUATION  Running_under_VMS -
                "Do you wish to run this program under VMS [Y/N]: "
$       IF .NOT. Running_under_VMS THEN GOTO Compile_LAB_1_C
$ !
$       SET VERIFY
$         CC /LIST LAB_1_C + ELN$:VAXELNC /LIBRARY
$         LINK LAB_1_C, SYS$INPUT /OPTION
SYS$SHARE:VAXCRTL/SHARE
$       GOTO Switch_off_verify
$ !
$ Compile_LAB_1_C:
$       SET VERIFY
$               CC -
                /LIST -
                /DEBUG LAB_1_C + ELN$:VAXELNC /LIBRARY
$ Link_LAB_1_C:
$               LINK -
                /DEBUG LAB_1_C -
                , ELN$:CRTLSHARE /LIBRARY -
                ,      RTLSHARE  /LIBRARY -
                ,      RTL       /LIBRARY
$       SET NOVERIFY
$ System_build:
$               @ELN_EBUILD_1   LAB_1_C
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
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
```

K.4 SOLUTION TO EXERCISE 2 - LAB_2.PAS

```
{------------------------------------------------------------------------
SOURCE:         LAB_2.PAS

PURPOSE:        To calculate temperatures on one of four scales:
                  1) Celsius      (ice melts: 0.0, water boils 100.0)
                  2) Fahrenheit   ( 32.0, 212.0)
                  3) Kelvin       (273.0, 373.0)
                  4) Reaumur      (  0.0,  80.0)
                given an input on one of these scales.
                A subprocess is created to perform the conversion.

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_2

LINK:           $ LINK /DEBUG LAB_2, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT LAB_2

NOTES:          1) Command procedure LAB_2.COM compiles, links
                   and builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Lab_2 [IDENT ('V1.000')];


CONST
        LF      = ''(10);   {line-feed}

VAR
        User_continues  : BOOLEAN := TRUE;

        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Sub_proc_exit_status,
        Status_returned : INTEGER;

        Answer,
        Temp_scale_in,
        Temp_scale_out  : CHAR;

        Process_ID      : PROCESS;

```

```
{-------------- FUNCTION DECLARATIONS --------------- }

FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;

FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
    END;

{---------------------------------------------------- }

FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Kelvin_to_Celsius := T - 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Reaumur_to_Celsius := T * 1.25;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Fahrenheit := T * 1.8 + 32.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Kelvin := T + 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Reaumur := T * 0.8;
    END;

{------------- END FUNCTION DECLARATIONS ------------- }
```

```

PROCESS_BLOCK Temperature_server;

BEGIN

CASE Temp_scale_in OF
        'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );

        'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );

        'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );

        'C' : Temp := Temp_value_in;
END;  {CASE Temp_scale_in}

{ provide conversion }

CASE Temp_scale_out OF
        'C'  : Temp_value_out := Temp;

        'F'  : Temp_value_out := Celsius_to_Fahrenheit ( Temp );

        'R'  : Temp_value_out := Celsius_to_Reaumur    ( Temp );

        'K'  : Temp_value_out := Celsius_to_Kelvin     ( Temp );
END; {CASE temp_scale_out}

END { end of PROCESS Temperature_server };

{------------------- PROGRAM BLOCK --------------------}

PROGRAM Lab_2 (OUTPUT);

   PROCEDURE Ensure_uppercase ( VAR C : CHAR );

   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};

BEGIN

```

```
WHILE User_continues DO

BEGIN
  WRITELN ( LF );

  WRITE  ( 'Please enter temperature: ' );
  READLN ( Temp_value_in );

  WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
  READLN ( Temp_scale_in );
  Ensure_uppercase ( Temp_scale_in );

  WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
  READLN ( Temp_scale_out );
  Ensure_uppercase ( Temp_scale_out );

CREATE_PROCESS ( Process_ID,
                 Temperature_server,
                 EXIT   := Sub_proc_exit_status,
                 STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);

WAIT_ANY ( Process_ID,
           STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);

IF NOT ODD ( Sub_proc_exit_status ) THEN
WRITELN ( 'Subprocess exit status was: ', Status_returned :1);

WRITELN (   'Temperature before conversion: ',
            Temp_value_in :6:1,
            ' deg',
            Temp_scale_in );

WRITELN (   'Temperature after conversion:  ',
            Temp_value_out :6:1,
            ' deg',
            Temp_scale_out );

WRITE  ( 'Do you wish to try again? [Y/N]: ' );
READLN (  Answer );

User_continues := Answer IN ['Y', 'y'];

END;

END {of PROGRAM}.
END {of MODULE };
```

K.4.1 Running LAB_2

```
$         ! LAB_2.COM
$         ! Command procedure to build the VAXELN
$         ! module LAB_2
$         !
$         ON ERROR THEN GOTO Switch_off_verify
$         SET DEFAULT Default_directory
$ !
$ Compile:
$                 @ELN_COMPILE_1  LAB_2
$ Link:
$                 @ELN_LINK_1     LAB_2
$ System_build:
$                 @ELN_EBUILD_1   LAB_2
$ Switch_off_verify:
$         SET NOVERIFY
$         EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
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
```

```
!
! 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

```
{-------------------------------------------------------------------
SOURCE:          LAB_2A.PAS

PURPOSE:         To calculate temperatures on one of four scales:
                   1) Celsius      (ice melts: 0.0, water boils 100.0)
                   2) Fahrenheit   ( 32.0, 212.0)
                   3) Kelvin       (273.0, 373.0)
                   4) Reaumur      (  0.0,  80.0)
                 given an input on one of these scales.
                 This version INCLUDEs the definitions of the temperature
                 conversion functions.

COMPILE:         $ EPASCAL /LIST /DEBUG LAB_2A_TDEFS
                 $ EPASCAL /LIST /DEBUG LAB_2A, LAB_2A_TDEFS /MODULE

LINK:            $ LINK /DEBUG LAB_2A, LAB_2A_TDEFS, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT LAB_2A

NOTES:           1) Command procedure LAB_2A.COM compiles, links
                    and builds this module into a VAXELN system
-------------------------------------------------------------------}
MODULE Lab_2A [IDENT ('V1.000')];

INCLUDE
        Lab_2A_Tdefs;  { declarations of temperature functions }

CONST
        LF      = ''(10);   {line-feed}

VAR
        User_continues  : BOOLEAN := TRUE;

        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Sub_proc_exit_status,
        Status_returned : INTEGER;

        Answer,
        Temp_scale_in,
        Temp_scale_out  : CHAR;

        Process_ID      : PROCESS;

```

```
{ -------------------------------------------------- }
PROCESS_BLOCK Temperature_server;
BEGIN
CASE Temp_scale_in OF
        'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
        'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );
        'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );
        'C' : Temp := Temp_value_in;
END;  {CASE Temp_scale_in}
{ provide conversion }
CASE Temp_scale_out OF
        'C'  : Temp_value_out := Temp;
        'F'  : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
        'R'  : Temp_value_out := Celsius_to_Reaumur    ( Temp );
        'K'  : Temp_value_out := Celsius_to_Kelvin     ( Temp );
END; {CASE temp_scale_out}
END { end of PROCESS Temperature_server };
```

```
{------------------ PROGRAM BLOCK -------------------}

PROGRAM Lab_2A (OUTPUT);

   PROCEDURE Ensure_uppercase ( VAR C : CHAR );

   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};

BEGIN

WHILE User_continues DO

BEGIN
  WRITELN ( LF );

  WRITE  ( 'Please enter temperature: ' );
  READLN ( Temp_value_in );

  WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
  READLN ( Temp_scale_in );
  Ensure_uppercase ( Temp_scale_in );

  WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
  READLN ( Temp_scale_out );
  Ensure_uppercase ( Temp_scale_out );

CREATE_PROCESS ( Process_ID,
                 Temperature_server,
                 EXIT   := Sub_proc_exit_status,
                 STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);

WAIT_ANY ( Process_ID,
           STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);

IF NOT ODD ( Sub_proc_exit_status ) THEN
WRITELN ( 'Subprocess exit status was: ', Status_returned :1);
```

```
WRITELN (   'Temperature before conversion: ',
            Temp_value_in :6:1,
            ' deg',
            Temp_scale_in );

WRITELN (   'Temperature after conversion:  ',
            Temp_value_out :6:1,
            ' deg',
            Temp_scale_out );

WRITE  ( 'Do you wish to try again? [Y/N]: ' );
READLN (  Answer );

User_continues := Answer IN ['Y', 'y'];

END;

END {of PROGRAM}.
END {of MODULE };
```

K.6 SOLUTION TO EXERCISE 2 - LAB_2A_TDEFS.PAS

The file of included definitions looks like this:

```
MODULE Lab_2A_Tdefs;

{ Temperature conversion functions }

FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;

FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
    END;

{------------------------------------------------------- }

FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Kelvin_to_Celsius := T - 273.0;
    END;

{------------------------------------------------------- }

FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Reaumur_to_Celsius := T * 1.25;
    END;

{------------------------------------------------------- }

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Fahrenheit := T * 1.8 + 32.0;
    END;

```

```
{---------------------------------------------------- }

FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Kelvin := T + 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Reaumur := T * 0.8;
    END;

{------------- END FUNCTION DECLARATIONS ------------- }

END.
```

K.6.1 Running LAB_2A

```
$        ! LAB_2A.COM
$        ! Command procedure to build the VAXELN
$        ! module LAB_2A
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$ Compile:
$        EPASCAL -
                 /LIST -
                 /DEBUG LAB_2A_TDEFS
$        EPASCAL -
                 /LIST -
                 /DEBUG LAB_2A, LAB_2A_TDEFS /MODULE
$ Link:
$        LINK -
                 /DEBUG LAB_2A, LAB_2A_TDEFS -
                 ,ELN$:RTLSHARE /LIBRARY -
                 ,RTL /LIBRARY
$        SET NOVERIFY
$ System_build:
$                @ELN_EBUILD_1   LAB_2A
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
1       characteristic /noconsole
2       program LAB_2A /debug
```

K.7 SOLUTION TO EXERCISE 2 - LAB_2B.PAS

This version INCLUDEs the declarations and constants.

```
{---------------------------------------------------------------------
SOURCE:         LAB_2B.PAS

PURPOSE:        To calculate temperatures on one of four scales:
                  1) Celsius      (ice melts: 0.0, water boils 100.0)
                  2) Fahrenheit   ( 32.0, 212.0)
                  3) Kelvin       (273.0, 373.0)
                  4) Reaumur      (  0.0,  80.0)
                given an input on one of these scales.
                This version uses a subprocess to perform the
                conversions and INCLUDEs the definitions of the
                temperature conversion functions, constants,
                variables and process block.

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_2B_DEFS
                $ EPASCAL /LIST /DEBUG LAB_2B, LAB_2B_DEFS /MODULE

LINK:           $ LINK /DEBUG LAB_2B, LAB_2B_DEFS, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT LAB_2B

NOTES:          1) Command procedure LAB_2B.COM compiles, links
                   and builds this module into a VAXELN system
--------------------------------------------------------------------}
MODULE Circuit_01 [IDENT ('V1.000')];

INCLUDE
        LAB_2B_DEFS;  { declarations of constants, variables,
                        temperature functions, and process block }

```

```
{------------------ PROGRAM BLOCK -------------------}

PROGRAM Create_job_01 (OUTPUT);

   PROCEDURE Ensure_uppercase ( VAR C : CHAR );

   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};

BEGIN

WHILE User_continues DO

BEGIN
  WRITELN ( LF );

  WRITE  ( 'Please enter temperature: ' );
  READLN ( Temp_value_in );

  WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
  READLN ( Temp_scale_in );
  Ensure_uppercase ( Temp_scale_in );

  WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
  READLN ( Temp_scale_out );
  Ensure_uppercase ( Temp_scale_out );

CREATE_PROCESS ( Process_ID,
                 Temperature_server,
                 EXIT   := Sub_proc_exit_status,
                 STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_PROCESS status was: ', Status_returned :1);

WAIT_ANY ( Process_ID,
           STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);

IF NOT ODD ( Sub_proc_exit_status ) THEN
WRITELN ( 'Subprocess exit status was: ', Status_returned :1);
```

```

WRITELN (   'Temperature before conversion: ',
            Temp_value_in :6:1,
            ' deg',
            Temp_scale_in );

WRITELN (   'Temperature after conversion:  ',
            Temp_value_out :6:1,
            ' deg',
            Temp_scale_out );

WRITE  ( 'Do you wish to try again? [Y/N]: ' );
READLN (  Answer );

User_continues := Answer IN ['Y', 'y'];

END;

END {of PROGRAM}.
END {of MODULE };
```

K.8 SOLUTION TO EXERCISE 2 - LAB_2B_DEFS.PAS

The file of included definitions looks like this:

```
MODULE LAB_2B_DEFS;

{ Temperature conversion functions }

INCLUDE
        LAB_2A_TDEFS;

CONST
        LF      = ''(10);   {line-feed}

VAR
        User_continues  : BOOLEAN := TRUE;

        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Sub_proc_exit_status,
        Status_returned : INTEGER;

        Answer,
        Temp_scale_in,
        Temp_scale_out  : CHAR;

        Process_ID      : PROCESS;

{ -------------------------------------------------- }

```

```
PROCESS_BLOCK Temperature_server;
BEGIN
CASE Temp_scale_in OF
        'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );
        'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );
        'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );
        'C' : Temp := Temp_value_in;
END;  {CASE Temp_scale_in}
{ provide conversion }
CASE Temp_scale_out OF
        'C'  : Temp_value_out := Temp;
        'F'  : Temp_value_out := Celsius_to_Fahrenheit ( Temp );
        'R'  : Temp_value_out := Celsius_to_Reaumur    ( Temp );
        'K'  : Temp_value_out := Celsius_to_Kelvin     ( Temp );
END; {CASE temp_scale_out}
END { end of PROCESS Temperature_server };
END { of MODULE }.
```

K.8.1 Running LAB_2B

```
$       ! LAB_2B.COM
$       ! Command procedure to build the VAXELN
$       ! module LAB_2B
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile:
$       EPASCAL -
                /LIST -
                /DEBUG LAB_2A_TDEFS
$       EPASCAL -
                /LIST -
                /DEBUG LAB_2B_DEFS, LAB_2A_TDEFS /MODULE
$       EPASCAL -
                /LIST -
                /DEBUG LAB_2B, LAB_2B_DEFS /MODULE
$ Link:
$       LINK -
                /DEBUG LAB_2B, LAB_2B_DEFS, LAB_2A_TDEFS -
                ,ELN$:RTLSHARE /LIBRARY -
                ,RTL /LIBRARY
$       SET NOVERIFY
$ System_build:
$               @ELN_EBUILD_1   LAB_2B
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program LAB_2B /debug
```

K.9 SOLUTION TO EXERCISE 2 - LAB_2_C.C

```
/* -----------------------------------------------------------------------
 * SOURCE:               LAB_2_C.C
 *
 * PURPOSE:         To calculate temperatures on one of four scales:
 *                   1) Celsius      (ice melts: 0.0, water boils 100.0)
 *                   2) Fahrenheit   ( 32.0, 212.0)
 *                   3) Kelvin       (273.0, 373.0)
 *                   4) Reaumur      (  0.0,  80.0)
 *                   given an input on one of these scales.
 *                  A subprocess is created to perform the conversion
 *
 * COMPILE:       $ CC /LIST /DEBUG LAB_2_C + ELN$:VAXELNC /LIBRARY
 *
 * LINK:          $ LINK /DEBUG LAB_2_C, -
 *               _$ ELN$:CRTLSHARE /LIBRARY, -
 *               _$ ELN$:RTLSHARE /LIBRARY, -
 *               _$ ELN$:RTL /LIBRARY
 *
 * BUILD:         $ EBUILD /NOEDIT LAB_2_C
 *
 * NOTES:       1) Command procedure LAB_2_C.COM compiles, links
 *                 and builds this module into a VAXELN system or
 *                 for running under VMS
 * -----------------------------------------------------------------------
 */

#include $vaxelnc
#include ctype          /* for the toupper function */

BOOLEAN user_continues = TRUE;

float   temp,
        temp_value_in,
        temp_value_out;

char    answer,
        temp_scale_in,
        temp_scale_out;

PROCESS process_id;

int     status_returned;

```

```
lab_2 ()
        MAIN_PROGRAM
        {
        float   fahrenheit_to_celsius(),
                kelvin_to_celsius(),
                reaumur_to_celsius(),
                celsius_to_fahrenheit(),
                celsius_to_kelvin(),
                celsius_to_reaumur();

        void    temperature_server();

        char    dummy;

while (user_continues)
    {
    printf ("\n\n");
    printf ("Please enter temperature: ");
    scanf  ("%f", &temp_value_in);

    dummy = getchar();     /* dispose of end-of-line */
    printf ("\nPlease enter scale for that temperature [K,C,F,R]: ");
    scanf  ("%c",  &temp_scale_in);

/*  uppercase temp_scale_in as necessary
 */
    temp_scale_in = toupper (temp_scale_in);

    dummy = getchar();     /* dispose of end-of-line */
    printf ("Please enter the conversion required [K,C,F,R]: ");
    scanf  ("%c",  &temp_scale_out);

/*  uppercase temp_scale_out as necessary
 */
    temp_scale_out = toupper (temp_scale_out);

    dummy = getchar();     /* dispose of end-of-line */

ker$create_process (&status_returned,
                    &process_id,
                    temperature_server,
                    NULL);
      if (! (status_returned & 1))
      printf ("KER$CREATE_PROCESS status was: \%d\n", status_returned);

```

```
ker$wait_any (&status_returned,
              NULL,
              NULL,
              process_id);
        if (! (status_returned & 1))
        printf ("KER$WAIT_ANY status was: \%d\n", status_returned);

printf ("Temperature before conversion: %6.1f deg%c\n",
        temp_value_in, temp_scale_in);
printf ("Temperature after conversion:  %6.1f deg%c\n",
        temp_value_out, temp_scale_out);

printf ("Do you wish to try again? [Y/N]: ");
scanf  ("%c", &answer);

user_continues = (answer == 'Y' || answer == 'y');

    } /* while user continues */

}

/*    ========== Temperature conversion functions ==========
 */

float fahrenheit_to_celsius (t)
float t;
        {
        return ((t-32.0) * 5.0 / 9.0);
        }

/* --------------------------------------------------- */

float kelvin_to_celsius (t)
float t;
        {
        return (t - 273.0);
        }

/* --------------------------------------------------- */

float reaumur_to_celsius (t)
float t;
        {
        return (t * 1.25);
        }

```

```
/* --------------------------------------------------- */

float celsius_to_fahrenheit (t)
float t;
        {
        return (t * 1.8 + 32.0);
        }

/* --------------------------------------------------- */

float celsius_to_kelvin (t)
float t;
        {
        return (t + 273.0);
        }

/* --------------------------------------------------- */

float celsius_to_reaumur (t)
float t;
        {
        return (t * 0.8);
        }

```

```
/* --------------------------------------------------
                  <Subprocess block>
   --------------------------------------------------
 */

void temperature_server()
{
switch (temp_scale_in )
   {
   case 'F': temp = fahrenheit_to_celsius (temp_value_in);
             break;
   case 'R': temp = reaumur_to_celsius    (temp_value_in);
             break;
   case 'K': temp = kelvin_to_celsius     (temp_value_in);
             break;
   case 'C': temp = temp_value_in;
             break;
   } /* switch temp_scale_in */

switch (temp_scale_out)
   {
   case 'C': temp_value_out = temp;
             break;
   case 'F': temp_value_out = celsius_to_fahrenheit (temp);
             break;
   case 'R': temp_value_out = celsius_to_reaumur (temp);
             break;
   case 'K': temp_value_out = celsius_to_kelvin (temp);
             break;
   } /* switch temp_scale_out */

return;
}
```

K.9.1 Running LAB_2_C

```
$       ! LAB_2_C.COM
$       ! Command procedure to build the VAXELN module LAB_2_C
$       ! into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$       SET VERIFY
$ !
$ Compile_LAB_2_C:
$               CC -
                /LIST -
                /DEBUG LAB_2_C + ELN$:VAXELNC /LIBRARY
$ Link_LAB_2_C:
$               LINK -
                /DEBUG LAB_2_C -
                , ELN$:CRTLSHARE /LIBRARY -
                ,      RTLSHARE  /LIBRARY -
                ,      RTL       /LIBRARY
$       SET NOVERIFY
$ System_build:
$               @ELN_EBUILD_1   LAB_2_C
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file looks like this:

```
characteristic /noconsole
program LAB_2_C /debug
```

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

```
{-----------------------------------------------------------------------
SOURCE:          LAB_3.PAS

PURPOSE:         To calculate temperatures on one of four scales:
                   1) Celsius      (ice melts: 0.0, water boils 100.0)
                   2) Fahrenheit   ( 32.0, 212.0)
                   3) Kelvin       (273.0, 373.0)
                   4) Reaumur      (  0.0,  80.0)
                 given an input on one of these scales.
                 This program creates a job and passes to it, via
                 arguments, the temperature conversion information.
                 The new job performs the conversion and prints the
                 results.

COMPILE:         $ EPASCAL /LIST /DEBUG LAB_3

LINK:            $ LINK /DEBUG LAB_3, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY

BUILD:           $ EBUILD /NOEDIT LAB_3

NOTES:           1) Command procedure LAB_3.COM compiles, links
                    and builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Lab_3 [IDENT ('V1.000')];

CONST
        LF      = ''(10);   {line-feed}
        Strlen  = 10;
VAR
        User_continues  : BOOLEAN := TRUE;

        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Status_returned : INTEGER;

        Answer,
        Temp_scale_in,
        Temp_scale_out  : CHAR;

        Temp_in_string  : VARYING_STRING(Strlen);

```

```
{------------------- PROGRAM BLOCK --------------------}

PROGRAM Lab_3 (OUTPUT);

VAR
        This_jobs_port,
        New_jobs_port    : PORT;

   PROCEDURE Ensure_uppercase ( VAR C : CHAR );

   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};

BEGIN

WHILE User_continues DO

BEGIN
  WRITELN ( LF );

  WRITE  ( 'Please enter temperature: ' );
  READLN ( Temp_value_in );

  WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
  READLN ( Temp_scale_in );
  Ensure_uppercase ( Temp_scale_in );

  WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
  READLN ( Temp_scale_out );
  Ensure_uppercase ( Temp_scale_out );

Temp_in_string := CONVERT ( VARYING_STRING(Strlen), Temp_value_in );

{ VAXELN seems to prefer a fresh port for each termination message -
  re-using the same port for exit messages causes the WAIT_ANY below
  to complete immediately }

CREATE_PORT ( This_jobs_port,
              STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_PORT status was: ', Status_returned :1);
```

```
CREATE_JOB ( New_jobs_port,
             'LAB_3A',
             '',
             '',
             Temp_in_string,
             Temp_scale_in,
             Temp_scale_out,
             NOTIFY := This_jobs_port,
             STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'CREATE_JOB status was: ', Status_returned :1);

WAIT_ANY ( This_jobs_port,
           STATUS := Status_returned );
        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'WAIT_ANY status was: ', Status_returned :1);

{ get rid of old PORT object to conserve system space }

DELETE ( This_jobs_port,
         STATUS := Status_returned );

        IF NOT ODD ( Status_returned ) THEN
        WRITELN ( 'DELETE status was: ', Status_returned :1);

WRITE  ( 'Do you wish to try again? [Y/N]: ' );
READLN (  Answer );

User_continues := Answer IN ['Y', 'y'];

END;

END {of PROGRAM}.
END {of MODULE };
```

K.11 SOLUTION TO EXERCISE 3 - LAB_3A.PAS

This is the code run by the job created from LAB_3.

```
{-------------------------------------------------------------------------
SOURCE:         LAB_3A.PAS

PURPOSE:        This program is run from LAB_3 and does the temperature
                conversions and displays the results

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_3A

LINK:           $ LINK /DEBUG LAB_3A, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT LAB_3

NOTES:          1) Command procedure LAB_3.COM compiles, links
                   and builds this module into a VAXELN system
------------------------------------------------------------------------}
MODULE Lab_3a [IDENT ('V1.000')];

CONST
        LF      = ''(10);   {line-feed}

VAR
        Temp,
        Temp_value_in,
        Temp_value_out  : REAL;

        Status_returned : INTEGER;

        Temp_scale_in,
        Temp_scale_out  : CHAR;

        Temp_in_string  : VARYING_STRING(10);

```

```
{--------------- FUNCTION DECLARATIONS --------------- }

FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;

FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
    END;

{---------------------------------------------------- }

FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Kelvin_to_Celsius := T - 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Reaumur_to_Celsius := T * 1.25;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Fahrenheit := T * 1.8 + 32.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Kelvin := T + 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Reaumur := T * 0.8;
    END;

{------------ END FUNCTION DECLARATIONS ------------ }

```

```
{------------------ PROGRAM BLOCK -------------------}

PROGRAM Lab_3a (OUTPUT);

CONST
        LF = ''(10);    { line-feed }

BEGIN

Temp_in_string := PROGRAM_ARGUMENT ( 3 );
Temp_scale_in  := PROGRAM_ARGUMENT ( 4 );
Temp_scale_out := PROGRAM_ARGUMENT ( 5 );

Temp_value_in  := CONVERT ( REAL, Temp_in_string );

CASE Temp_scale_in OF
        'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );

        'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );

        'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );

        'C' : Temp := Temp_value_in;
END;  {CASE Temp_scale_in}

{ provide conversion }

CASE Temp_scale_out OF
        'C'  : Temp_value_out := Temp;

        'F'  : Temp_value_out := Celsius_to_Fahrenheit ( Temp );

        'R'  : Temp_value_out := Celsius_to_Reaumur    ( Temp );

        'K'  : Temp_value_out := Celsius_to_Kelvin     ( Temp );
END; {CASE temp_scale_out}

WRITELN ( LF, LF );

WRITELN (    'Temperature before conversion: ',
             Temp_value_in :6:1,
             ' deg',
             Temp_scale_in );

WRITELN (    'Temperature after conversion:  ',
             Temp_value_out :6:1,
             ' deg',
             Temp_scale_out );

END {of PROGRAM}.
END {of MODULE };
```

K.11.1 Running LAB_3

```
$        ! LAB_3.COM
$        ! Command procedure to build the VAXELN
$        ! modules LAB_3 and LAB_3A
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$ Compile:
$                @ELN_COMPILE_1  LAB_3
$                @ELN_COMPILE_1  LAB_3A
$ Link:
$                @ELN_LINK_1     LAB_3
$                @ELN_LINK_1     LAB_3A
$ System_build:
$                @ELN_EBUILD_1   LAB_3
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nonetwork /nofile /noserver
program LAB_3 /debug
program LAB_3A /norun /debug
```

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.

K.12 SOLUTION TO EXERCISE 4 - LAB_4.PAS

```
{------------------------------------------------------------------------
SOURCE:         LAB_4.PAS

PURPOSE:        To demonstrate creation and access to a VAXELN AREA

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_4, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG LAB_4, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT LAB_4

NOTES:          1) Command procedure LAB_4.COM compiles, links and
                   builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Lab_4 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        Limit                   = 100;
        Name_of_area            = 'Celsius_table';

{ define the area with its preamble }
TYPE
        Temperature_table = RECORD
           First_value  : INTEGER;
           Last_value   : INTEGER;
           Scale_name   : CHAR;
           Scale_wanted : CHAR;
           Temperatures : ARRAY [1..Limit] OF REAL;
        END;

VAR
        Area_identity           : AREA;
        This_port,
        New_jobs_port           : PORT;
        Start_of_area           : ^Temperature_table;
        Returned_status         : INTEGER;

{---------------------------------------------}

```

```
PROGRAM Lab_4;

VAR
        I                  : 1..Limit;

BEGIN

CREATE_AREA ( Area_identity,
              Start_of_area,
              Name_of_area,
              STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_AREA' );

WITH Start_of_area^ DO

BEGIN
  First_value   := 1;
  Last_value    := First_value + Limit - 1;
  Scale_name    := 'C';
  Scale_wanted := 'F';

  FOR I := First_value TO Last_value DO Temperatures[I] := I * 1.0;
END;

CREATE_PORT ( This_port,
              STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_PORT' );

CREATE_JOB  ( New_jobs_port,
              'LAB_4A',
              '',
              '',
              Name_of_area,
              NOTIFY  := This_port,
              STATUS  := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_JOB' );

WAIT_ANY ( This_port,
           STATUS := returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

END { of PROGRAM}.
END { of MODULE };
```

K.13 SOLUTION TO EXERCISE 4 - LAB_4A.PAS

This is the code run by the job created from LAB_4.

```
{--------------------------------------------------------------------
SOURCE:         LAB_4A.PAS

PURPOSE:        This program is run from LAB_4 and does the temperature
                conversions and displays the results

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_4A

LINK:           $ LINK /DEBUG LAB_4A, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT LAB_4

NOTES:          1) Command procedure LAB_4.COM compiles, links
                   and builds this module into a VAXELN system
-------------------------------------------------------------------}
MODULE Lab_4a [IDENT ('V1.000')];

CONST
        LF      = ''(10);   {line-feed}
        Limit   = 100;

TYPE
        Temperature_table = RECORD
           First_value  : INTEGER;
           Last_value   : INTEGER;
           Scale_name   : CHAR;
           Scale_wanted : CHAR;
           Temperatures : ARRAY [1..Limit] OF REAL;
        END;

VAR
        Area_ID              : AREA;
        This_port,
        New_jobs_port        : PORT;
        Name_of_area         : VARYING_STRING(31);
        Start_of_area        : ^Temperature_table;

        Returned_status      : INTEGER;

{ Temperature conversion function definitions here... }

        %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'

{------------------- PROGRAM BLOCK --------------------}

```

```
PROGRAM Lab_4a (OUTPUT);

VAR
        I : INTEGER;

BEGIN

Name_of_area        := PROGRAM_ARGUMENT ( 3 );

CREATE_AREA (   Area_ID,
                Start_of_area,
                Name_of_area,
                STATUS := Returned_status );
        IF NOT ODD ( Returned_status ) THEN
        WRITELN ( 'CREATE_AREA status was: ', Returned_status :1 );

WITH Start_of_area^ DO

BEGIN

CASE Scale_name OF

        'F', 'f' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Fahrenheit_to_Celsius ( Temperatures[I] );

        'R', 'r' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Reaumur_to_Celsius ( Temperatures[I] );

        'K', 'k' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Kelvin_to_Celsius ( Temperatures[I] );

END; {CASE Scale_name}

CASE Scale_wanted OF

        'F', 'f' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Celsius_to_Fahrenheit ( Temperatures[I] );

        'R', 'r' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Celsius_to_Reaumur ( Temperatures[I] );

        'K', 'k' :  FOR I := First_value TO Last_value DO
                        Temperatures[I] :=
                          Celsius_to_Kelvin ( Temperatures[I] );

END; {CASE Scale_wanted}
```

```

END; {WITH}

CREATE_PORT ( This_port,
              STATUS := Returned_status );

        IF NOT ODD ( Returned_status ) THEN
        WRITELN ( 'CREATE_PORT status was: ', Returned_status :1 );

CREATE_JOB  ( New_jobs_port,
              'LAB_4B',
              '',
              '',
              Name_of_area,
              NOTIFY  := This_port,
              STATUS  := Returned_status );

        IF NOT ODD ( Returned_status ) THEN
        WRITELN ( 'CREATE_JOB status was: ', Returned_status :1 );

WAIT_ANY ( This_port,
           STATUS := returned_status );

        IF NOT ODD ( Returned_status ) THEN
        WRITELN ( 'WAIT_ANY status was: ', Returned_status :1 );

END {of PROGRAM}.
END {of MODULE };
```

The file included in the compilation of LAB_4A looks like this:

```
{-------------- FUNCTION DECLARATIONS -------------- }

FUNCTION Temperature_conversion ( T : REAL ) :REAL; FUNCTION_TYPE;

FUNCTION Fahrenheit_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Fahrenheit_to_Celsius := (T - 32.0) * 5.0 / 9.0;
    END;

{---------------------------------------------------- }

FUNCTION Kelvin_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Kelvin_to_Celsius := T - 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Reaumur_to_Celsius OF TYPE Temperature_conversion;

    BEGIN
        Reaumur_to_Celsius := T * 1.25;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Fahrenheit OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Fahrenheit := T * 1.8 + 32.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Kelvin OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Kelvin := T + 273.0;
    END;

{---------------------------------------------------- }

FUNCTION Celsius_to_Reaumur OF TYPE Temperature_conversion;

    BEGIN
        Celsius_to_Reaumur := T * 0.8;
    END;

{------------ END FUNCTION DECLARATIONS ------------ }
```

K.14 SOLUTION TO EXERCISE 4 - LAB_4B.PAS

This is the code run by the job created from LAB_4A.

```
{-----------------------------------------------------------------------
SOURCE:         LAB_4B.PAS

PURPOSE:        This program is run from LAB_4A and displays the results
                of the temperature conversion

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_4B

LINK:           $ LINK /DEBUG LAB_4B, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY

BUILD:          $ EBUILD /NOEDIT LAB_4

NOTES:          1) Command procedure LAB_4.COM compiles, links
                   and builds this module into a VAXELN system
----------------------------------------------------------------------}
MODULE Lab_4b [IDENT ('V1.000')];

CONST
        LF      = ''(10);   {line-feed}
        Limit   = 100;

TYPE
        Temperature_table = RECORD
           First_value   : INTEGER;
           Last_value    : INTEGER;
           Scale_name    : CHAR;
           Scale_wanted  : CHAR;
           Temperatures  : ARRAY [1..Limit] OF REAL;
        END;

VAR
        Area_ID             : AREA;
        Name_of_area        : VARYING_STRING(31);
        Start_of_area       : ^Temperature_table;
        Full_scale_name     : VARYING_STRING(10); {size of Fahrenheit}
        Returned_status     : INTEGER;

{------------------- PROGRAM BLOCK --------------------}

PROGRAM Lab_4b (OUTPUT);

VAR
        I : INTEGER;

```

```
BEGIN
Name_of_area        := PROGRAM_ARGUMENT ( 3 );
CREATE_AREA (   Area_ID,
                Start_of_area,
                Name_of_area,
                STATUS := Returned_status );
        IF NOT ODD ( Returned_status ) THEN
        WRITELN ( 'CREATE_AREA status was: ', Returned_status :1 );
WITH Start_of_area^ DO
BEGIN
   CASE Scale_wanted OF
        'C', 'c'        : Full_scale_name := 'Celsius';
        'F', 'f'        : Full_scale_name := 'Fahrenheit';
        'K', 'k'        : Full_scale_name := 'Kelvin';
        'R', 'r'        : Full_scale_name := 'Reaumur';
   END; {CASE}
   WRITELN ( 'Table of temperatures in ', Full_scale_name );
   WRITELN ( 'From original values ', First_value:1,
             ' to ', Last_value:1, ' in degrees ', Scale_name, LF );
   FOR I := First_value TO Last_value DO
    BEGIN
        IF ( (I-1) MOD 10 ) = 0 THEN WRITELN;
        WRITE ( Temperatures[I]:7:1 );
    END;
END {WITH Start_of_area};
END {of PROGRAM}.
END {of MODULE };
```

K.14.1 Running LAB_4

```
$        ! LAB_4.COM
$        ! Command procedure to build the VAXELN
$        ! modules LAB_4, LAB_4A and LAB_4B
$        !
$        ON ERROR THEN GOTO Switch_off_verify
$        SET DEFAULT Default_directory
$ !
$        SET VERIFY
$        EPASCAL -
                 /DEBUG -
                 /LIST LAB_4, VAXELN-MODULES /LIBRARY
$        LINK -
                 /DEBUG LAB_4, VAXELN-MODULES /LIBRARY -
                 ,ELN$:RTLSHARE /LIBRARY -
                 ,RTL /LIBRARY
$        SET NOVERIFY
$ !
$                @ELN_COMPILE_1  LAB_4A
$                @ELN_COMPILE_1  LAB_4B
$                @ELN_LINK_1     LAB_4A
$                @ELN_LINK_1     LAB_4B
$ System_build:
$                @ELN_EBUILD_1   LAB_4
$ Switch_off_verify:
$        SET NOVERIFY
$        EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole /nofile
program LAB_4 /debug
program LAB_4A /norun /debug
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
```

```
! 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:

```
<------------------------- 11  BYTES ------------------------->

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 10  |   9 |   8 |   7 |   6 |   5 |   4 |   3 |   2 |   1 |   0 |
|     |     |     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|<--- Temp_value_in --->|<--- Temp_value_out -->|  ^     ^     ^
                                                   |     |     |
                           Temp_scale_in  ----+     |     |
                           Temp_scale_out ----------+     |
                           Message_status ----------------+
```

Packet as defined for LAB_5A:

```
<------------------------- 11  BYTES ------------------------->

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |     |     |     |
| 10  |   9 |   8 |   7 |   6 |   5 |   4 |   3 |   2 |   1 |   0 |
|     |     |     |     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|<--- Temp_value_out -->|<--- Temp_value_in --->|  ^     ^     ^
                                                   |     |     |
                           Temp_scale_out ----+     |     |
                           Temp_scale_in  ----------+     |
                           Message_status ----------------+
```

```
{-------------------------------------------------------------------
SOURCE:         LAB_5.PAS

PURPOSE:        To demonstrate creation of a circuit between VAXELN jobs
                and two-way communication through it.
                This program prompts for temperature information
                then passes a request to LAB_5A for conversion and
                return of the results via the circuit established
                by LAB_5A

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_5, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG LAB_5, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT LAB_5

NOTES:          1) Command procedure LAB_5.COM compiles, links
                   and builds this module into a VAXELN system
-------------------------------------------------------------------}
MODULE Lab_5 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        LF      = ''(10);     {line-feed}
TYPE
        Message_kind        = ( Continue, Finished );

        Message_type        = RECORD
            Message_status   : Message_kind;
            Temp_scale_out,
            Temp_scale_in    : CHAR;
            Temp_value_out,
            Temp_value_in    : REAL;
        END;

```

```
{------------------- PROGRAM BLOCK -------------------}

PROGRAM Lab_5 (OUTPUT);

VAR
        Answer                 : CHAR;
        User_continues         : BOOLEAN := TRUE;
        My_new_port            : PORT;
        Outgoing_message,
        Incoming_message       : ^Message_type;
        Message_identity       : MESSAGE;
        My_new_port_name       : NAME;
        Returned_status        : INTEGER;
        Temperature_sent       : REAL;
        Scale_sent             : CHAR;

   PROCEDURE Ensure_uppercase ( VAR C : CHAR );

   BEGIN
    IF C IN ['a'..'z'] THEN C := CHR ( ORD (C) - 32 );
   END {PROCEDURE Ensure_uppercase};

{ MAIN program start }

BEGIN

CREATE_PORT (    My_new_port,
                 STATUS := Returned_status );

             Check_status_and_report ( Returned_status,
                                       'CREATE_PORT' );

CREATE_NAME (    My_new_port_name,
                 'MASTER_PORT',
                 My_new_port,
                 TABLE  := NAME$UNIVERSAL,
                 STATUS := Returned_status );

             Check_status_and_report ( Returned_status,
                                       'CREATE_NAME' );

INITIALIZATION_DONE ( STATUS := Returned_status );

             Check_status_and_report ( Returned_status,
                                       'INITIALIZATION_DONE' );

ACCEPT_CIRCUIT ( My_new_port,
                  STATUS := Returned_status );

             Check_status_and_report ( Returned_status,
                                       'ACCEPT_CIRCUIT' );
```

```
WHILE User_continues DO

BEGIN

CREATE_MESSAGE   (    Message_identity,
                      Outgoing_message,
                      STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

{ Initialize the message packet and put 'continue' flag in status.
  Preserve outgoing temp and scale values as they are 'lost' from
  outgoing message packet once sent (message pointers are undefined
  after a SEND) }

WITH Outgoing_message^ DO

   BEGIN
   Message_status := Continue;

   WRITELN ( LF );
    WRITE  ( 'Please enter temperature: ' );
    READLN ( Temp_value_out );
    Temperature_sent := Temp_value_out;

   WRITE  ( 'Please enter scale for that temperature [K,C,F,R]: ');
    READLN ( Temp_scale_out );
    Ensure_uppercase ( Temp_scale_out );
    Scale_sent := Temp_scale_out;

   WRITE  ( 'Please enter the conversion required [K,C,F,R]: ');
   READLN ( Temp_scale_in );
   Ensure_uppercase ( Temp_scale_in );
   END;

SEND             (    Message_identity,
                      My_new_port,
                      STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );
```

```
WAIT_ANY ( My_new_port );

RECEIVE  (  Message_identity,
            Incoming_message,
            My_new_port,
            STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'RECEIVE' );

WRITELN (   'Temperature before conversion: ',
            Temperature_sent :6:1,
            ' deg',
            Scale_sent );

WITH Incoming_message^ DO
        WRITELN (   'Temperature after conversion:  ',
                    Temp_value_in :6:1,
                    ' deg',
                    Temp_scale_in );

        WRITE  ( 'Do you wish to try again? [Y/N]: ' );
        READLN (  Answer );

        User_continues := Answer IN ['Y', 'y'];

END;

CREATE_MESSAGE  (   Message_identity,
                    Outgoing_message,
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_MESSAGE' );

{ now that the user has finished put 'finished flag in status }

WITH Outgoing_message^ DO
   Message_status := Finished;

SEND            (   Message_identity,
                    My_new_port,
                    STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );

END {of PROGRAM}.
END {of MODULE };
```

K.16 SOLUTION TO EXERCISE 5 - LAB_5A.PAS

This program connects the circuit with LAB_5

```
{-------------------------------------------------------------------------
SOURCE:         LAB_5A.PAS

PURPOSE:        To demonstrate circuit connection with another job
                This program connects a circuit with LAB_5 and
                through that circuit receives conversion requests.
                After servicing these requests the results are returned
                on the same circuit.

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_5A, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG LAB_5A, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT LAB_5

NOTES:          1) Command procedure LAB_5.COM compiles, links
                   and builds this module into a VAXELN system
-----------------------------------------------------------------------}
MODULE Lab_5a [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

TYPE
        Message_kind        = ( Continue, Finished );

        Message_type        = RECORD
            Message_status   : Message_kind;
            Temp_scale_in,
            Temp_scale_out   : CHAR;
            Temp_value_in,
            Temp_value_out   : REAL;
        END;

```

```
{-------------------- PROGRAM BLOCK --------------------}

PROGRAM Lab_5a (OUTPUT);

VAR
        Process_has_exited     : BOOLEAN := FALSE;
        Count_of_arguments,
        Returned_status         : INTEGER;
        Temp                    : REAL;
        Incoming_message,
        Outgoing_message        : ^Message_type;
        Message_identity        : MESSAGE;
        New_port                : PORT;

{ include function definitions }

        %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'

{ MAIN program start }

BEGIN

CREATE_PORT     (    New_port,
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CREATE_PORT' );

CONNECT_CIRCUIT (    New_port,
                     DESTINATION_NAME := 'MASTER_PORT',
                     STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'CONNECT_CIRCUIT' );

REPEAT

  WAIT_ANY ( New_port );

  RECEIVE  (  Message_identity,
              Incoming_message,
              New_port,
              STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'RECEIVE' );

```

```
{ NOTE: Process exits here when message contains FINISHED flag }

  IF Incoming_message^.Message_status = Finished THEN EXIT;

  WITH Incoming_message^ DO
    BEGIN
        CASE Temp_scale_in OF
           'F' : Temp := Fahrenheit_to_Celsius ( Temp_value_in );

           'R' : Temp := Reaumur_to_Celsius    ( Temp_value_in );

           'K' : Temp := Kelvin_to_Celsius     ( Temp_value_in );

           'C' : Temp := Temp_value_in;
        END;  {CASE Temp_scale_in}
    END;

  CREATE_MESSAGE         (   Message_identity,
                             Outgoing_message,
                             STATUS := Returned_status );

          Check_status_and_report ( Returned_status,
                                    'CREATE_MESSAGE' );

  WITH Outgoing_message^ DO
    BEGIN
        CASE Incoming_message^.Temp_scale_out OF
           'C'  : BEGIN
                  Temp_value_out := Temp;
                  Temp_scale_out := 'C';
                  END;
           'F'  : BEGIN
                  Temp_value_out := Celsius_to_Fahrenheit ( Temp );
                  temp_scale_out := 'F';
                  END;
           'R'  : BEGIN
                  Temp_value_out := Celsius_to_Reaumur    ( Temp );
                  Temp_scale_out := 'R';
                  END;
           'K'  : BEGIN
                  Temp_value_out := Celsius_to_Kelvin     ( Temp );
                  Temp_scale_out := 'K';
                  END;
        END; {CASE temp_scale_out}
    END;
```

```
SEND             (    Message_identity,
                      New_port,
                      STATUS := Returned_status );

            Check_status_and_report ( Returned_status,
                                      'SEND' );

UNTIL Process_has_exited; { Never satisifies this statement }

END {of PROGRAM}.
END {of MODULE };
```

K.16.1 Running LAB_5

```
$       ! LAB_5.COM
$       ! Command procedure to build the VAXELN modules
$       ! LAB_5 and LAB_5A into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile1:
$       EPASCAL -
                /LIST -
                /DEBUG LAB_5, VAXELN-MODULES /LIBRARY
$ Compile2:
$       EPASCAL -
                /LIST -
                /DEBUG LAB_5A, VAXELN-MODULES /LIBRARY
$ Link1:
$       LINK -
                /DEBUG LAB_5 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link2:
$       LINK -
                /DEBUG LAB_5A -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT LAB_5
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program LAB_5 /initialize /debug
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
```

K.17 SOLUTION TO EXERCISE 6 - LAB_6.PAS

```
{---------------------------------------------------------------------------
SOURCE:          LAB_6.PAS

PURPOSE:         This program reads a file of temperatures from a VAX
                 host into an AREA.
                 The program then hands over to LAB_6A which converts
                 those temperatures to a chosen scale and outputs the
                 results.

COMPILE:         $ EPASCAL /LIST /DEBUG LAB_6, -
                 _$  VAXELN-MODULES /LIBRARY

LINK:            $ LINK /DEBUG LAB_6, VAXELN-MODULES /LIBRARY, -
                 _$ ELN$:RTLSHARE /LIBRARY, -
                 _$ ELN$:RTL /LIBRARY /INCLUDE= -
                 _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:           $ EBUILD /NOEDIT LAB_6

NOTES:           1) Command procedure LAB_6.COM compiles, links
                    and builds this module into a VAXELN system
                 2) Change node and file spec as appropriate
---------------------------------------------------------------------------}
MODULE Lab_6 [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        First_year      = 1659;
        Last_year       = 2058;

TYPE
        Months     = (January,   February, March,    April,
                      May,       June,     July,     August,
                      September, October,  November, December,
                      Annual);

```

```
        Temperature_scale = (Celsius, Fahrenheit, Kelvin, Reaumur);

        Years_record = RECORD
          Year_number   : First_year..Last_year;
          Monthly_temps : ARRAY [January..Annual] OF INTEGER;
        END;

        Temperature_table = RECORD
          Latest_year  : First_year..Last_year;
          Scale_name   : Temperature_scale;
          Scale_wanted : Temperature_scale;
          Temperatures : ARRAY [First_year..Last_year] OF Years_record;
        END;

VAR
        Temperature_area: AREA;
        Year             : First_year..Last_year;
        Start_of_temps   : ^Temperature_table;
        Cet              : TEXT;
        This_port,
        LAB_6As_port     : PORT;
        Returned_status  : INTEGER;


PROGRAM Lab_6 (INPUT, OUTPUT);

VAR
        I                : First_year..Last_year;
        Period           : Months;

BEGIN

{ this open statement won't work unless you are on node 1.241 and
  the device and directory spec are correct }

OPEN ( Cet,
       FILE_NAME := '1.241::DISK$COURSEDSK:[SHONE]TEMPS.DAT',
       HISTORY   := HISTORY$OLD);

RESET (Cet);

CREATE_AREA (   Temperature_area,
                Start_of_temps,
                'ENGLISH_TEMPERATURES',
                STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_AREA' );
```

```
I := First_year;

WHILE NOT EOF(CET) DO
BEGIN

  WITH Start_of_temps^ DO
   BEGIN
     READ (Cet, Temperatures[I].Year_number );
     FOR Period := January TO Annual DO
         READ(Cet, Temperatures[I].Monthly_temps[Period]);
     READLN(Cet);
     I := I + 1;
   END;

END;

CLOSE (Cet);

WITH Start_of_temps^ DO

BEGIN
  Latest_year  := Temperatures[I-1].Year_number;
  Scale_name   := Celsius;
  Scale_wanted := Fahrenheit;
END;

JOB_PORT   ( This_port,
             STATUS   := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'JOB_PORT' );

CREATE_JOB ( LAB_6As_port,
             'LAB_6A',
             '',
             '',
             'ENGLISH_TEMPERATURES',
             NOTIFY   := This_port,
             STATUS   := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_JOB' );

WAIT_ANY ( This_port,
           STATUS   := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'WAIT_ANY' );

END {of PROGRAM}.
END {of MODULE};
```

K.18 SOLUTION TO EXERCISE 6 - LAB_6A.PAS

This program connects the circuit with LAB_6

```
{----------------------------------------------------------------------
SOURCE:         LAB_6A.PAS

PURPOSE:        This program converts temperatures in an AREA created
                by LAB_6 to a chosen scale and prints the results

COMPILE:        $ EPASCAL /LIST /DEBUG LAB_6A, -
                _$  VAXELN-MODULES /LIBRARY

LINK:           $ LINK /DEBUG LAB_6A, VAXELN-MODULES /LIBRARY, -
                _$ ELN$:RTLSHARE /LIBRARY, -
                _$ ELN$:RTL /LIBRARY /INCLUDE= -
                _$ (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                _$  OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)

BUILD:          $ EBUILD /NOEDIT LAB_6

NOTES:          1) Command procedure LAB_6.COM compiles, links
                   and builds this module into a VAXELN system
---------------------------------------------------------------------}
MODULE Lab_6A [IDENT ('V1.000')];

INCLUDE
        Check_status_and_report;

CONST
        LF              = ''(10);       {line-feed}
        First_year      = 1659;
        Last_year       = 2058;

TYPE
        Months     = (January,   February, March,    April,
                      May,       June,     July,     August,
                      September, October,  November, December,
                      Annual);

```

```
        Temperature_scale = (Celsius, Fahrenheit, Kelvin, Reaumur);

        Years_record = RECORD
          Year_number   : First_year..Last_year;
          Monthly_temps : ARRAY [January..Annual] OF INTEGER;
        END;

        Temperature_table = RECORD
          Latest_year  : First_year..Last_year;
          Scale_name   : Temperature_scale;
          Scale_wanted : Temperature_scale;
          Temperatures : ARRAY [First_year..Last_year] OF Years_record;
        END;

VAR
        Temperature_area: AREA;
        Year            : First_year..Last_year;
        Start_of_temps  : ^Temperature_table;
        Returned_status : INTEGER;

{ Temperature conversion function definitions here }

        %INCLUDE 'LAB_4A_FUNC_DEFS.INC/LIST'

{ ------------------- PROGRAM BLOCK ------------------- }

PROGRAM Lab_6A (OUTPUT);

VAR
        I       : First_year..Last_year;
        Period  : January..Annual;

BEGIN

CREATE_AREA (   Temperature_area,
                Start_of_temps,
                'ENGLISH_TEMPERATURES',
                STATUS := Returned_status );

        Check_status_and_report ( Returned_status,
                                  'CREATE_AREA' );
WITH Start_of_temps^ DO
BEGIN
  WRITELN ( ' Temperatures for Central England in degrees ',
          Scale_wanted, LF );
  WRITELN ( ' Period: ', First_year:4, ' to ', Latest_year:4,
          LF, LF);

```

```
    FOR I := First_year TO Latest_year DO

    BEGIN
     WRITE ( Temperatures[I].Year_number:5 );
     FOR Period := January TO Annual DO
       CASE Scale_wanted OF

          Fahrenheit :
             WRITE ( Celsius_to_Fahrenheit
                     (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);

          Kelvin     :
             WRITE ( Celsius_to_Kelvin
                     (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);

          Reaumur    :
             WRITE ( Celsius_to_Reaumur
                     (Temperatures[I].Monthly_temps[Period] * 0.1):5:1);

       END; {CASE}

      WRITELN;

  END; {FOR loop}
  END; {WITH}

  END {of PROGRAM}.
  END {of MODULE};
```

K.18.1 Running LAB_6

```
$       ! LAB_6.COM
$       ! Command procedure to build the VAXELN modules
$       ! LAB_6 and LAB_6A into a system
$       !
$       ON ERROR THEN GOTO Switch_off_verify
$       SET DEFAULT Default_directory
$ !
$       SET VERIFY
$ Compile1:
$       EPASCAL -
                /LIST -
                /DEBUG LAB_6, VAXELN-MODULES /LIBRARY
$ Compile2:
$       EPASCAL -
                /LIST -
                /DEBUG LAB_6A, VAXELN-MODULES /LIBRARY
$ Link1:
$       LINK -
                /DEBUG LAB_6 -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ Link2:
$       LINK -
                /DEBUG LAB_6A -
                ,VAXELN-MODULES  /LIBRARY -
                ,ELN$:RTLSHARE   /LIBRARY -
                ,ELN$:RTL /LIBRARY /INCLUDE= -
                (ELN$MSGDEF_TEXT, KER$MSGDEF_TEXT, -
                 OTS$MSGDEF_TEXT, PAS$MSGDEF_TEXT)
$ System_build:
$       EBUILD -
                        /NOEDIT LAB_6
$ Switch_off_verify:
$       SET NOVERIFY
$       EXIT
```

The .DAT file used by EBUILD looks like this:

```
characteristic /noconsole
program LAB_6 /debug
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