Supermax System V

Running Interpreted Pascal

Dansk Data Elektronik A/S
15. Sep 1987

Version 1.1

Table of Contents.

	Page
1. Introduction	1.2
2. The Compiler	2.1
2.1 Running the Compiler	2.1
2.2 Compile Time Options	2.3
3. Running the Interpreter	3.1
4. Environment File - External Procedures	4.1
4.1 Contents of Environment File	4.1
4.2 The Program Envredit	4.2
5. The program ppc	5.1
6. Implementation Size Limits	6.1
Appendix A. Compile Time Error Messages	a.1
Appendix B. Run Time (Interpreter) Error Messages	b.1
Appendix C. Necessary files	c.1

1. Introduction.

This manual describes the use of the compiler and interpreter for Supermax Interpreted Pascal. In addition the contents of environment files is described. An environment file may contain external procedures and functions. See the Pascal C Interface Manual for writing external procedures and functions and how to link environment files.

Dansk Data Elektronik A/S reserves the right to change the specifications in this manual without warning. Dansk Data Elektronik A/S is not responsible for the effects of typographical errors and other inaccuracies in this manual, and cannot be held liable for the effect of the implementation and use of the structures described herein.

2. The Compiler.

2.1 Running the Compiler.

The compiler is located in the file /pbin/pascal. The compiler checks if the environment TARGETMC is set to 68000 or 68020. If not pascal terminates without compiling. Otherwise pascal uses the value of the environment when generating code for calling the interpreter. If TARGETMC is 68000 code for calling /etc/inter will be generated otherwise code for calling /etc/inter20 will be generated.

The compiler may execute in either parameter mode or in dialog mode.

When +q is given as parameter the compiler asks questions (user input shown underlined):

```
$ pascal +q
Source code file:          tst.p
P-code file:               tst
Environment file:          tst.j
List option (t/l/ /q):     l
List file:                 tst.list
Conditions:                d
```

Source code file:
The source code file is the file containing the source code of the Pascal program. It will normally (but not necessarily) be of type p.

P-code file:
The p-code file is the file in which the translated program should be stored. It must be typeless.

Environment file:
The environment file is the file in which the source code for a possible environment file (see section 4) will be stored. It will normally be of type j.

If the end-of-file key is pressed in reply to this request in dialog mode, the file will not be written, and in this case it is not allowed to declare external procedures and functions in the Pascal program.

List option:
The listoption may be used to control the list output produced by the compiler. The listoptions recognized by the compiler are T (t), L (l), and Q (q). The listoption may be changed during a compilation using the L and Q compile time options (see section 2.2).

If L, l, T, or t is specified as the listoption, and the compiler is executed in dialog mode the compiler will ask for the name of the file in which to store a source listing of the compiled program.

If L (l) is specified as the listoption, the compiler will produce a source listing of the compiled program including the number of each source line compiled, as well as the block level (the number of RECORDs, BEGINs, REPEATs, and statement CASEs minus the number of ENDs, UNTILs, and OTHERWISEs processed) before the line is compiled. Error messages will appear in the listfile just after the line in which the error was detected.

If T (t) is specified as the listoption, the compiler will produce the same output as mentioned under L above, but after each subroutine the compiler will output information about the size of the compiled subroutine (number of bytes of p-code generated), accumulated segment size (bytes of p-code), and size (in bytes) of the temporary stack area that will be allocated for the subroutine's variables, local parameters etc. Further, after each record declaration the size of the record will be output.

The Q (q) listoption suppresses all output except error messages.

If an illegal listoption is specified or if no listoption is specified, the compiler will use the output device to keep the operator informed about the progress of the compilation by displaying the source line number and the name of the procedure currently being compiled plus a dot for each line compiled. Error messages will appear on the error device together with a printout of the line in which the error was detected.

Conditions:
Conditions is simply a number of capital letters that control the conditional compilation, see section 2.2. If the program contains compiler directives such as (*$XH*).....(*$X-*), the code between the two compiler directives will be compiled only if the letter H is among the letters given in the conditions parameter.

The parameter mode corresponding to the dialog mode shown in the start of the section is:

```
$ pascal -i tst.p -o tst -e tst.j -Ll -T tst.list -CD
```

The default values are:

```
for pcode  - the name of the source code file with .p omitted
for envr   - external procedures and functions not allowed
for listop - blank
for cond   - no conditions
```

if the given listoption implies a listing the default name of the list file is the name of the source code file with .p substituted by .list.

Compilation may be aborted at any time by entering a ctrl c at the terminal that was used to start the compilation. Using ctrl c will not kill other Pascal programs running in the background on the same terminal, and no trace dump will be produced.

It should be noted that SPC/1 and Supermax p-code are not compatible.

2.2 Compile Time Options.

The compilers may be instructed to generate code according to certain options; in particular, it may be requested to insert or omit run-time test instructions, and it may be requested to include files. Compiler directive are written as comments and are designated as such by a $-character as the first character of the comment:

```
(*$<option sequence>*)
```

The option sequence is a sequence of instructions separated by commas. Each instruction consists of a letter, designating the option, followed either by a plus (+) if the option is to be activated or a minus (-) if the action is to be passivated, or by a digit, or by a filename. Example:

```
(*$C+,Ideclfile*)
```

Illegal syntax in a compiler directive is not reported by the compiler, but the results are unpredictable.

The following options are currently available:

B causes the compiler to generate line numbers referring to the lines in the listing of the program produced by the compiler and not (as default) relative to the source code file, the line originates from. (Also see option D)

C causes the compiler to generate I/O check instructions after each statement which performs any I/O. The instruction checks to see if the I/O operation was accomplished successfully. In the case of an unsuccessful I/O operation the program will be terminated with a Supermax I/O error message. Note that no I/O check instructions are generated after RESET, REWRITE, LOCK or CHAIN procedure calls.

When automatic I/O checking is off, the user must check the value of the IORESULT function after each I/O operation.

C+ I/O check instructions are generated (default).
C- I/O check instructions are not generated.

D causes the compiler to generate line numbers in the p-code. If a run time error occurs the interpreter will print the number of the source line corresponding to the code that was executed when the error occurred, along with the name of the file containing the source code. Each line number generated occupies 2 or 3 bytes of p-code. The generated line number refers to the line number in the file (maybe an included file) the line originates from (also see option J).

D+ line numbers are generated (default).
D- line numbers are not generated.

E causes the compiler to generate the names of all declared external subroutines in the type j file. The names are generated in the declaration order.

G determines whether Pascal GOTO statements are allowed within the program. This option may be used to restrict novice programmers from using the GOTO statement in situations where structured constructs like FOR, WHILE, REPEAT, and CASE statements would be more appropriate

G+ allows the use of the GOTO statement.
G- causes the compiler to generate a syntax error upon encountering a GOTO statement (default).

I includes a source file into the compilation. The characters between 'I' and the terminating '*)' are taken as the file name of the source file to be included. The comment must be closed at the end of the file name, therefore no other options can follow the file name.

Example: (*$D-,I/usr/an/tst.p*)

The compiler cannot keep track of nested inclusions, that is, an included file may not have an include file compiler directive. This will result in a fatal syntax error. If nested inclusions are wanted, the Pascal pre-processor can be used. Note that files included by $I are included during the compilation, while files included by the pre-processor are included before compilation.

J causes the compiler to generate code keeping track of which source code file the currently executed subroutine originates from, thus making it possible to give an exact message about the location of a run-time error. As this information takes up much space it is recommended that this facility is only used as long as the program is erroneous.

J+ file names are generated (default).
J- file names are not generated.

L controls whether the compiler will generate a program listing of the following source text. This directive is analogous to the L listoption discussed in section 2.1. The default value of this option is set by the listoption when the compiler is started.

L+ start output of source listing on the list file.
L- stop output of source listing.

M causes the compiler to output the number of bytes remaining in the heap of the compiler. This number gives the user a hint as to whether a compiler run time error Stack or heap overflow may be expected if the program is expanded. The heap grows with each declaration of an identifier.

N This letter must be followed by two digits. These digits are taken as an integer, and the number of lines per page in a program listing is set to this number.

For example (*$N30*)

O this option controls whether or not the p-code generated should be optimized.

O+ turns optimizing on.
O- turns optimizing off (default).

P causes the compiler to skip to a new page on the printer if the compiler is generating a source listing on the printer at the time when the P compiler directive is encountered.

Q is the 'quiet compiler' option which can be used to suppress the output to the output device of procedure names, line numbers, and dots detailing the progress of the compilation. This compiler directive is analogous to the Q listoption discussed in section 2.1. The default value of this option is set by the listoption when the compiler is started.

Q+ causes the compiler to suppress output to the output device.
Q- causes the compiler to output procedure names, line numbers, and dots to the output device.

R This option controls whether or not the compiler should output additional code to perform checking on array and string subscripts and assignments to variables of subrange types.

R+ turns range checking on (default).
R- turns range checking off.

Programs compiled with the R option set will run slightly faster and require less code; however, if an invalid index occurs or an invalid assignment is made, the program will not be terminated with a run time error. Until a program has been completely tested and is known to be correct, it is usually best to compile with the R+ option set. Note that certain string indexing errors (index<0 or >255) are detected even if range checking is disabled.

S This option makes it possible to use arbitrary ascii characters in strings and as character constants. S must be followed by a character. This character is denoted an escape character. In strings and character constants this escape character can be given followed by two hexadecimal digits. The two hexadecimal digits specify an ascii character. When a character no longer should be used as escape character the escape character can be deleted by the compile time option S followed by -. Default escape character is '-'. Example:

```
(*$S\*)
writeln('error\07');  (* writes error and beeps *)
(*$S-*)
```

T causes the compiler to generate line numbers in the p-code. Contrary to compile time option D only line numbers referring to the first line in a block (main program or subroutine) are generated, thus enabling a subroutine trace during execution if the debug flag in the environment file is set. Compile time option T is activated only if D is not used.

```
   T- line numbers are not generated (default)
D-,T+ line numbers are generated
```

U,V both options are generated by the Pascal Pre-processor making the compiler able to print correct line numbers and filenames on discovering syntax errors. These options should not be used by the programmer.

X specifies code which is to be compiled only under certain conditions. This directive takes the form (*$Xn*), where n is either a capital letter or a minus sign. If n is a capital letter the following code is compiled only if that letter was given in the conditions parameter when the compiler was started (see section 2.1). If n is a minus sign the following code is compiled unconditionally. This conditional compilation facility is useful if two almost identical versions of a program are desired; for example, certain statements producing test output may be conditionally compiled so that it is easy to switch from a test version of a program to a non-test version. Conditional compilation can also be obtained using Pre-processor commands.

Example:

Assume that the following program resides in the file tst.p:

```
PROGRAM TEST;
BEGIN
  WRITELN('COMMON');
  (*$XT*) WRITELN('TTT'); (*$X-*)
  WRITELN('COMMON');
  (*$XQ*) WRITELN('QQQ'); (*$X-*)
  WRITELN('COMMON');
END.
```

If this program is compiled in the following manner:

```
$ pascal -i tst.p
```

it will be equivalent with the following program:

```
PROGRAM TEST;
BEGIN
  WRITELN('COMMON');
  WRITELN('COMMON');
  WRITELN('COMMON');
END.
```

If the program is compiled as follows:

```
$ pascal -i tst.p -CQ
```

it will be equivalent with the following program:

```
PROGRAM TEST;
BEGIN
  WRITELN('COMMON');
  WRITELN('COMMON');
  WRITELN('QQQ');
  WRITELN('COMMON');
END.
```

If, finally, the program is compiled this way:

```
$ pascal -i tst.p -CQT
```

it will be equivalent with the following program:

```
PROGRAM TEST;
BEGIN
  WRITELN('COMMON');
  WRITELN('TTT');
  WRITELN('COMMON');
  WRITELN('QQQ');
  WRITELN('COMMON');
END.
```

3. Running the Interpreter.

There exist two Pascal interpreters. They are located in /etc/inter and /etc/inter20 and are used to execute the p-code output from the Pascal Compiler. The pascal compiler generates information in the p-code about which interpreter should be called.

The difference between the two interpreters lies in what instructions are used to perform operations on the type REAL. Supermax Pascal can run on both mc68000 and mc68020 processors. /etc/inter can be used on both processors while /etc/inter20 only can run on a mc68020 processor as it utilizes this processors floating point co-processor.

To ensure that programs are compiled correctly for use on either a mc68000 processor or a mc68020 processor a special environment TARGETMC must be set. TARGETMC can have the value 68000 or 68020.

The interpreters are executed:

```
$ filename parameters
```

or

```
$ /etc/inter filename parameters (or /etc/inter20 filename parameters)
```

Filename is the name of the file containing the p-code and parameters are the parameters passed to the Pascal program.

When a Pascal program is executed the interpreter searches for the p-code file using the users PATH environment.

The interpreter uses segments 4, 5, 6, 7, 8, 9, and 10. Segment 5 is data area for variables, segment 6 p-code area and segment 7 heap area. Segment 4 is used in connection with writing p-code in memory. Segment 8 is the environment file and segment 9 is data area for the environment file. Segment 10 is reserved for the use of text libraries (see Supermax Pascal User's Guide section 2).

The version of the interpreter is written when a run time error occurs; it is also possible to get the version of the interpreter by executing:

```
$ what /etc/inter
```

4. Environment File - External Procedures.

In connection with the execution of a Pascal program an environment file may be supplied. This file is really a load module produced from

- an assembly language code originating from the compiler (the type j file) and possibly modified
- a set of subroutines written in C, being the external procedures and functions of the Pascal program

The environment file is linked to segment 8 and 9.

4.1 Contents of Environment File.

An environment file contains the following:

```
load information

size of data area for variables
size of p-code area
size of heap
debug flag

address of external procedure number 1, zero for none
address of external procedure number 2, zero for none
  .
  .
address of external procedure number 128, zero for none

the external procedures, if any
```

The size of data area for variables, the size of p-code area and the size of heap control the data area allocation performed prior to the execution of the Pascal program. (The Pascal compiler generates the size of P-code in the type j file).

The debug flag enables the Pascal debugger, which outputs a line number on the error device each time a new line is executed.

The external procedures and functions are numbered in the order the first calls of the respective routines occur in the programtext. The maximum number of external subroutines allowed is 128.

When the Pascal interpreter is executed it looks for an environment file with the same name as the p-code file but of type e. If such a file is found, it is loaded and its contents are used when executing the program. Otherwise the following default environment is used: 8 K bytes each for p-code and variables, no heap, not debug and no external subroutines.

An environment file without external subroutines can be created by the program /pbin/envredit. See the manual Pascal C interface how to link an environment file with external procedures and functions.

4.2 The Program Envredit.

The program envredit creates an environment file or changes an existing one. The information that can be set or changed by envredit is:

- size of data area for variables
- size of p-code area
- size of heap
- debug flag

An environment file created by envredit cannot contain external subroutines.

Envredit runs in dialog or parameter mode. When +q is given as parameter the program asks questions. Example:

```
$ envredit +q
Enter unit name: tst.e
Enter size of data area for variable (hex): a000
Enter size of p-code area (hex): 10000
Enter size of heap (hex): 200
Enter debug flag: 1
```

The corresponding parameter mode is:

```
$ envredit -i tst.e -d a000 -p 10000 -h 200 -D 1
```

If the environment file does not exist it is created, otherwise it is updated.

5. The program ppc.

ppc is a program which depending on what types of files and what options are given executes one or more of the programs ppp, pascal and pasenvr. ppc is an abbreviation of Pascal to Pcode Compiler.

ppc will check if the environment TARGETMC is set to 68000 or 68020. If not ppc terminates without compiling. Otherwise ppc uses the value of the environment in selecting which programs to be run and which standard libraries to use when linking the environment file.

ppc works as follows:

- a file of type p is sent through the preprocessor before output from the preprocessor is sent on to pascal.

- a file of type pp is given directly to pascal.

- if a file of type p and the option -P is given to ppc the file will be given directly to pascal.

- if a file of type j is given ppc will create an envrionment file by executing pasenvr.

The following options can be given to ppc:

-o after -o the name of the resulting file is given.

-L may be followed by l, L, t, T, q or space. If a space is written the preprocessor and the compiler will output a dot for each line. If q is written output will be suppressed and if l, L, t, or t is given pascal will produce a listing of the program. If -T option is not given the listing will be placed in <xxx>.list given the name of the sourcefile is <xxx>.p.

-T must be followed by a space and the name of the file in which the program listing should be placed if listoption t, T, l or L has been given.

-C the letters written following C are given to pascal as conditions.

-P type p files are given directly to pascal.

-v verbose: ppc writes to standard output what fase it is currently executing.

-O must be followed by names of type .o files separated by ;. These files are passed to pasenvr.

-l must be followed by names of libraries separated by ;. These libraries are passed to pasenvr.

-e an environment file will be created.

-j type j file will be generated.

Both the type j and e file will be generated if either -e, -O or -L is given.

Example:

```
$ ppc -v -o tst tst.p -O pwait.o;xor.o -l mylib.a -e -P  -CM -Ll
```

First ppc will send tst.p to pascal with the condition M and causing pascal to create a type j-file and a listing of the source text in tst.list. Next tst.j with be passed to pasenvr along with the type o files pwait.o and xor.o and the library mylib.a. The result is that the pcode file tst and the environment file tst.e are available.

6. Implementation Size Limits.

The following is a list of limitations imposed upon the user by the current implementation of Supermax Interpreted Pascal:

1) The maximum number of bytes of object code in a procedure or function is 4000. Local variables in a procedure or function can occupy a maximum of 32766 bytes of memory.

2) The maximum number of characters is 255 in a string variable and 32767 in a longstring and cstring variable.

3) The maximum number of segment procedures and segment functions is 15.

4) The maximum number of procedures or functions within a segment is 255.

5) The maximum number of external procedures is 128

6) The maximum size of p-code in memory for a program is 64 K bytes.

7) The standard function memavail does not work when more than 65534 bytes are available in the heap.

8) The maximum number of bytes of global variables is 32000.

Appendix A. Compile Time Error Messages.

Errors with numbers > 400 cause the compiler to terminate.

```
  1: error in simple type
  2: identifier expected
  4: ')' expected
  5: ':' or '..' expected
  6: symbol illegal in context (may be missing ';' on the line  above
     or ';' in front of ELSE)
  7: error in parameter list
  8: 'OF' expected
  9: '(' expected
 10: error in type
 11: '(.' expected
 12: '.)' expected
 13: 'END' expected
 14: ';' expected
 15: integer expected
 16: '=' expected
 17: 'BEGIN' expected
 18: error in declaration part
 19: error in <field list>
 20: ',' expected
 21: '.' expected

 50: error in constant
 51: ':=' expected
 52: 'THEN' expected
 53: 'UNTIL' expected
 54: 'DO' expected
 55: 'TO' or 'DOWNTO' expected in FOR-statement
 56: 'EXITIF' expected
 57: 'ENDLOOP' expected
 58: error in <factor> (bad expression)
 59: error in variable
```

101: identifier declared twice
102: low bound exceeds high bound
103: identifier is not of the appropriate class (may be a type identifier used where a variable is required)
104: undeclared identifier
105: sign not allowed
106: number expected
107: incompatible subrange types
108: file not allowed here
109: type must not be real
110: <tagfield> type must be scalar or subrange
111: incompatible with <tagfield> part
113: index type must be a scalar or a subrange
114: base type must not be real
115: base type must be a scalar or a subrange
116: error in type of standard procedure parameter
117: unsatisfied forward reference
119: re-specified parameters not ok for a forward declared procedure
120: function result type must be scalar, subrange or pointer
121: file value parameter not allowed
122: a forward declared function's result type cannot be respecified
123: missing result type in function declaration
125: error in type of standard function parameter
126: number of parameters does not agree with declaration
127: illegal parameter substitution
128: result type does not agree with declaration
129: type conflict of operands
130: expression is not of set type
131: tests on equality allowed only
132: strict inclusion not allowed
133: file comparison not allowed
134: illegal type of operand(s)
135: type of operand must be boolean
136: set element type must be scalar or subrange
137: set element types must be compatible
138: type of variable is not array
139: index type is not compatible with the declaration
140: type of variable is not record
141: type of variable must be file or pointer
142: illegal parameter solution

143: illegal type of loop control variable
144: illegal type of expression
145: type conflict
146: assignment of files not allowed
147: label type incompatible with selecting expression
148: subrange bounds must be scalar
149: index type must not be integer
150: assignment to standard function is not allowed
152: no such field in this record
153: type error in read
154: actual parameter must be a variable
155: control variable cannot be formal or non-local
156: multidefined case label
158: no such variant in this record
159: real or string tagfields not allowed
160: previous declaration was not forward
161: again forward declared
162: parameter size must be constant
163: missing variant in declaration
165: multidefined label
166: multideclared label
167: undeclared label
168: undefined label
169: error in base set
173: externaloption not specified when compilation was started
174: parameter universal declared in non-external procedure
175: only files and unpacked records may be UNIV declared
176: parameter declared as cstring in non-external procedure
177: parameter of type power must be variable declared in external procedure declaration
178: comparison not allowed on cstring

180: constant index out of bounds
181: overflow in constant expression
182: division by constant zero
183: case constant too large

201: error in real number - digit expected
202: string constant must not exceed source line
203: integer constant exceeds range
204: illegal hexadecimal character

250: too many scopes of nested identifiers
251: too many nested procedures or functions - or too many procedures or functions in a segment
253: procedure too long
254: CASE statement to long
257: too many external procedures
258: location counter exceeds range

397: implementation restriction
398: implementation restriction
399: implementation restriction

400: illegal character in text
401: unexpected end of input
403: 'PROGRAM' expected
408: include control comment not allowed in inclusion file
409: error in parameters to the Pascal compiler omitted

10xxx: error during open of inclusion file
11xxx: error during open of source file
12xxx: error during create/open of p-code file
13xxx: error during create/open of environment file (type j)
14xxx: error during output to environment file
15xxx: error during output to P-code file
16xxx: error during input from source file
17xxx: error during seek in P-code file
18xxx: error during open of list file

In the error messages with numbers >= 10000 the last three digits represent the Supermax Operating System error code (see Supermax System Operation Guide Appendix A).

Appendix B. Run Time (Interpreter) Error Messages.

During interpretation of a Pascal program, the interpreter checks for a number of error conditions. If an error condition is detected it is reported in one of the following ways:

<error condition> near line nnnn
Inter version dd.mm.yyyy

or

I/O error zzz near line nnnn
Inter version dd.mm.yyyy

or

Pascal error: <error text>. Supermax errorcode zzz

where dd.mm.yyyy is the version date of the Pascal interpreter. nnnn is the line number of the last executed Pascal statement that was compiled with the (*$D+*) or (*$T+*) option (see section 2.2). If the (*$D-*) and the (*$T-*) option was in effect during the compilation of the program section executed before the error occured, nnnn will be 0000. <error condition> will be replaced by:

- Alphabet table used for string comparison erroneous
- Break by interrupt key
- Compilation of program not finished
- Division by zero
- External procedure error (probably wrong environment file, see chapter 4)
- Long to short string overflow
- Long to short integer overflow
- Index error (invalid index or range of variable exceeded)
- Illegal string assignment
- Stack or heap overflow (change the size of data area or heap in environment file see section 4.2)
- String too long or parameter error in intrinsic procedure

- Standard procedure not implemented
- Floating point overflow
- Floating point error (error in PWROFTEN call)
- Non-existent segment called (system error)
- More than 255 characters in record in sequential file
- Illegal p-code instruction
- Exitting procedure never called

If the Pascal compiler fails with Floating point overflow or Floating point error the source statement just after the last one listed on the list device probably contains a (real) constant which is not acceptable to the compiler.

An I/O error code appears when an I/O operation fails, and I/O checking has not been disabled through a (*$C-*) compiler option. zzz is the error code. The Supermax I/O error codes are explained in Supermax System Operation Guide Appendix A.

A Pascal error: <error text> can only appear before the interpretation of the Pascal program starts or when a segment should be read. For instance it appears if the interpreter cannot open the p-code file or cannot create the partitions to be used when interpreting the Pascal program. The error is explained in <error text> and zzz is the Supermax error code. See Supermax System Operation Guide Appendix A.

Note that when one of the error types occurs, all open files will be closed, and if the files are sequential ones that have been opened with a REWRITE call, end-of-file will be at the current file position.

Appendix C. Necessary files.

The Interpreted Pascal System consists of the following files:

/pbin/pascal	- compiler
/etc/inter	- interpreter (mc68000)
/etc/inter20	- interpreter (mc68020)
/pbin/ppc	- control program for compilation
/pbin/ppp	- pascal pre-processor
/pbin/ptperror	- error messages for pascal
/pbin/extdecl.p	- declarations of external routines
/pbin/envredit	- creation/change of environment file

The following files are only used when linking environment files:

/pbin/pasenvr	- linking environment file (see the Pascal C Interface Manual)
/pbin/prt0.o /pbin/default.ld	} used when linking environment file
/lib/libpext.a	- library containing the standard external routines

Note: to be able to link environment files containing standard external subroutines the library:

/lib/libc.a (mc68000)
or
/lib20/libc.a (mc68020)

must be available.

Furthermore it is of course only possible to write and use your own external subroutines written in C if the appropriate C-compiler system is available.

Along with the Interpreted Pascal System the following files are supplied:

/nlslib/pascal/uk	- text file containing syntax error messages
/usr/lib/alphabet/dk	- table on the Danish alphabet used for alphabetic comparison on strings.