Pascal 180 External Reference Specification

M R Renfro

DISCLAIMER:

This document is an internal working paper and does not represent any intent on the part of Control Data Corporation.

REVISION RECORD	
REVISION	DESCRIPTION
A 83-06-06	Preliminary version.
B 84-04-30	Update for Analysis and Design Phase.
C 84-10-31	Update to resolve comments to Revision B.

1.0 INTRODUCTION AND OBJECTIVES

This document is the External Reference Specification for Pascal 180. It specifies the language implemented, the user interface to the compiler and the diagnostics produced by the compiler.

2.0 REFERENCES

2.0 REFERENCES

1. Specification for Computer Programming Language Pascal
   ISO dp7185, 1983.

2. American National Standard Pascal Computer Programming
   Language, ANSI/IEEE770x3.97-1983.

3. Pascal 180 Project Plan.
   M Renfro, DCS Log S4407.

4. Cyber 180 System Interface Standard [SIS]
   DCS Log ID S2196

5. Pascal Version 1 Reference Manual.
   CDC Publication 60497700, 1982.

6. Pascal 180 DR, DCS Log ID S4647.

3.0 FEATURE DESCRIPTION

3.0 FEATURE DESCRIPTION

3.1 ABSTRACT

Pascal 180 implements the programming language Pascal, as described below.

3.2 DESCRIPTION

Pascal 180 implements the Pascal language as defined by the ISO Standard and the ANSI Standard, with exceptions noted below. The ISO Standard provides the base language definition. Certain extensions, detailed below, are provided for compatibility with Cyber 170 Pascal. These extensions to the above standards can be flagged at the user's request.

3.2.1 EXTENSIONS

Extensions to the ISO Standard are listed below as additions to the ISO Standard [reference 1].

3.2.1.1 Non-alphanumeric Characters in Identifiers

Pascal identifers are extended to allow the underscore '_' and the currency symbol '$' as part of an identifier in the same manner as a digit.

Modify definition of identifier [p8, 6.1.3 of ISO Standard] as follows:

```
identifier = letter {letter|digit|letter-symbol} .

letter-symbol = "_" | "$" .
```

3.2.1.2 VALUE Declarations

The implementation of value declarations is defined as:

Modify definition of block [p12 of ISO Standard] as follows:

Insert after 'variable-declaration-part' in 'block =':

```
value-declaration-part
```

Insert after 'variable-declaration-part = ...':

```
value-declaration-part =
      ["value" value-declaration ";"
      {value-declaration ";"} ] .
```

3.0 FEATURE DESCRIPTION
3.2.1.2 VALUE Declarations

Insert after variable-declaration [p27-30 of ISO Standard] the following definitions:

```
value-declaration =
     variable-identifier "=" value-specification .
value-specification =
     constant ! "nil" ! set-value ! structured-value .
set-value = set-constructor .
structured-value =
     [type-identifier] structured-value-specification .
structured-value-specification =
     "(" structured-value-element
     {"," structured-value-element} ")" .
structured-value-element =
     [replication-factor] value-specification .
replication_factor =
     constant "of" .
```

3.2.1.3 OTHERWISE_Clause_in_CASE_Statement

The Implementation of the otherwise clause in the case statement is defined as:

Modify definition of case statement [p56 of ISO Standard] to read:

```
case statement =
     "case" case-index "of"
     case-list-element          {";"          case-list-element}
case-statement-tail .
```

Insert following definition after 'case-index':

```
case-statement-tail = ["otherwise" statement] [;] "end".
```

3.2.1.4 Additional_Predefined_Routines

In addition to those predefined routines required bye the standard, Pascal will supply CARD, CLOCK, DATE, EXPO, HALT, MESSAGE, TIME, and UNDEFINED. Descriptions of these routines will be supplied during the analysis and design phase.

3.2.2 IMPLEMENTATION DEFINED AND IMPLEMENTATION DEPENDENT FEATURES

Implementation defined and implementation dependent features of Pascal are addressed in both standards.

3.2.2.1 Implementation_Defined_Features

Implementation defined features may differ between processors, but will be defined by all processors. Below are the implementation

3.0 FEATURE DESCRIPTION
3.2.2.1 Implementation Defined Features

defined features of Pascal and the Pascal 180 implementation:

Implementation Defined Feature	Pascal 180 Implementation
Subset of characters which can occur in char/string literal	The ASCII character set.
The ordinal representation of char values.	The ASCII ordinals.
Subset of real numbers allowed.	-10e1232 thru -10e-1233 0 10e-1233 thru 10e1232
Value of maxint.	9223372036854775807
Number of characters for exponent field on output.	6, [E+dddd or E-dddd].
Symbol denoting exponent on output ('e' or 'E')	'E', upper case.
Case for Boolean values on output.	Upper case, [TRUE or FALSE]
Default values for total width on integer, Boolean and real output.	Integer: # digits+1 Boolean: 6 Real : 22 (applies only to floating point)
Point of actual action and checking of assertions on I/O routines.	Pre-assertions will be checked prior to execution of the code necessary to perform the specified I/O. Post-assertions will be true after the code necessary to perform the specified I/O action has been executed.

Effect of page on textfile.	page(f) is equivalent to:
	writeln(f);
	write(f,'1');

Effect of reset/rewrite to	Rewrite and reset have the

Input/output.	effect described in both standards section 6.6.5.2 except that after reset on interactive files, the file buffer variable will not contain the first character from the interactive file.
Binding of file type program parameters.	Bound at run time to external files. External file names are essociated by runtime parameters or have the same name as the program parameter.

3.2.2.2 Implementation Dependent Features

Implementation dependent features may differ between processors and need not be defined by any given processor. Below are the implementation dependent features of Pascal and the Pascal 180 implementation:

Implementation Dependent Feature	Pascal 180 Implementation
Order of evaluation of index expressions.	Left to right.
Order of evaluation of set member designators.	Left to right.
Order of evaluation of operands of dyadic operators.	Left to right, complete evaluation.
Order of evaluation, access and binding of actual parameters to functions and procedures.	Order of evaluation: left to right. Access is from the stack. Binding is at runtime.
For assignment statements, order of: evaluate access of variable, evaluate expression.	Evaluate expression, then evaluate variable access.

Effect of inspecting a textfile to which page has been applied	The character '1' will appear in the first column of the line to which the procedure page has been applied.
Binding of non-file type program parameters.	Bound at run time. Actual parameters must be SCL values. Type of formal parameters are restricted to integer, Boolean and string. Formal program parameters will be treated in the program as global variables. Missing actual parameters will result in no initialization of the corresponding program parameter.

3.2.3 ERRORS NOT DETECTED

The ISO Standard defines an error to be a violation of the standard (typically requiring execution of the program to detect) which the processor is not required to detect. Conformance to the standard does, however, require a documented list of errors not detected. This list will be found in section 5.4 .

3.3 INTERFACES

User interface to Pascal 180 will be SIS conforming. Below are listed the control statement parameters accepted by Pascal:

Parameter	Alias	Description
BINARY_OBJECT	B	Binary Object Code output file. B = <file> This parameter specifies the file to contain the object code produced by Pascal. B=$NULL indicates no object file is to be output. Single specified value parameter.

Default: B=$LOCAL.LGO

COMPILATION_DIRECTIVE CD — Compilation directive switch.

CD = <boolean>

This parameter determines if Pascal compiler directives are to be honored.

Single option parameter.
Default: CD=ON

DEBUG D — Debug option.

D = <option list>

This parameter specifies the debug options to be selected. Multiple options may be selected. Note that the options are negative and are used to deselect generation of debug code.

NC No checking. Do not generate code to check parameters to procedures and functions with an external directive.

NT No tables. Do not generate line number and symbol tables as part of the object code.

Multiple option parameter.
Default: D=NONE [i.e., parameter check code and tables will be generated.]

ERROR E — Error file.

E = <file>

This parameter specifies the file to which Pascal will write the text of diagnostics, of EL level or higher. Diagnostics are also written to the L file, if present. If E and L name the same file, only one copy of the diagnostic will be output.

E=$NULL results in no error file.

Single specified value parameter.
Default: E=$ERRORS

ERROR_LEVEL EL — Error level.

EL = <option>

This parameter indicates the severity level of diagnostic which will be output by Pascal.

W Warning level and higher diagnostics are output.

F Fatal level diagnostics only are output.

Single specified value parameter.
Default: EL=W

INPUT I — Input file.

I = <file>

This parameter specifies the source input file name to Pascal.

I=$NULL is invalid.

Single specified value parameter.
Default: I=$INPUT

LIST L — Listing file.

L = <file>

This parameter specifies the file to which Pascal will write the source listing, diagnostics, object listing, statistics and reference/attributes.

L=$NULL results in no listings output.

Single specified value parameter.

Default: L=$LIST

LIST_OPTIONS LO — Listing options.

LO = <option list>

The options of this parameter specify the information which is to appear on the L file. Multiple options may be specified.

A Attributes. A list of the attributes of each entity in the program. If option R is selected, the references are shown on the same listing.

O Object listing. A listing of the object program with assembler mnemonics.

R Cross reference listing. A listing which shows definition and uses of all entities of the program.

S Source listing. Source listing of the program.

LO=NONE causes no listing options to be selected.

Multiple option parameter.
Default: LO=S.

RUNTIME_CHECKS RC — Runtime checks code generation.

RC = <option list>

This parameter controls which runtime checks will be compiled into the object program. Multiple options may be selected.

F Files checking. Selects checking of errors involving file variables and buffer variables.

N Pointer checking. Selects checking

of misuse of pointer variables and invalid usage of new and dispose procedures.

R Range checks. Selects range checking for subrange and set assignments and case variables.

S Subscript checks. Selects array subscript bound checking.

T Tag field checks. Selects checking of accesses to variant records for consistency with their tag field (if one exists).

RC =ALL causes selection of all checks.

Multiple value specified parameter.
Default: RC=NONE

STANDARDS_DIAGNOSTICS SD Standards diagnostics.

SD = (level, standard)

This parameter specifies whether use of non-standard extensions in a program are to be diagnosed. The first option defines the error level to be assumed by such diagnostics and the second option determines which of the two standards is to apply.

Level:

W Standard errors result in warning errors.

F Standard errors result in fatal errors.

Standard:

ANSI The ANSI standard is the basis.

ISO The ISO standard is the basis.

SD = NONE causes standards errors not

to be diagnosed.

SD = F (or W) causes the ISO standard to be the basis.

SD = (,ISO) [or (,ANSI)] causes the level to be W.

Multiple specified value parameter.
Default: SD=NONE

STATUS — None — Status variable.

STATUS = <status-variable>

This parameter specifies the name of the SCL status variable to be set by Pascal to indicate the occurrence of error conditions.

Single specified value parameter.
Default: No status information is returned.

TERMINATION_ERROR_LEVEL — TEL — Termination error level.

TEL = <option>

This parameter indicates the severity level of diagnostic which will cause Pascal to return an abnormal STATUS.

W Warning level and higher diagnostics cause abnormal STATUS.

F Fatal level diagnostics only cause abnormal STATUS.

Single specified value parameter.
Default: TEL=F

3.4 ABORTS_AND_RECOVERY

3.4.1 COMPILE TIME

Detection of errors of EL or above (see section 3.3) will result in setting the STATUS variable. Action taken at that point is a user responsibility.

3.4.2 RUN TIME

Detection of an error at run time will result in the output of a diagnostic message, raising of an exception condition and termination of execution. If Debug is in force, the user will have that facility available.

3.5 PERFORMANCE

Performance specifications are provided in the Pascal 180 Design Requirements [reference 6].

4.0 PRODUCT-LEVEL DESCRIPTION

Procedure and function calls in Pascal 180 are implemented in accordance with the SIS.

5.0 ERRORS

5.0 ERRORS

5.1 FATAL_DIAGNOSTICS_(COMPILE_TIME)

Pascal 180 will provide the following compile time diagnostics. Texts are listed below. These diagnostics are unnumbered and will be output to the source listing (as well as the error file) following the source line which contained the error. Format of the diagnostics will be as follows:

FATAL *ERROR* {column number} text

Diagnostic Texts
{information in brackets is variable}

A conformant array's index must be of some ordinal type.
A file type may not be another file or a structured type with a component of type file.
A subrange's lower bound must be less than or equal to its upper bound.
ABS built-in function can be applied only to integers or reals.
All conformant array actual parameters in a section must be of the same type.
An active FOR control variable cannot be assigned to or passed as a VAR parameter.
An active FOR control variable cannot be read into.
An array subscript must be of some ordinal type.
An array's index must be of some ordinal type.
Argument number {integer} to READ is not assignment compatible with the component type of the file variable.
Argument number {integer} to WRITE is not assignment compatible with the component type of the file variable.
Argument of REWRITE must be a file.
Argument of {procedure identifier} must be a file.
Argument to EOF or EOLN must be a file.
Argument to PAGE must be a file.
Base of target set type is not compatible with base of value set type.
Built-in function {function identifier} can only be called in expressions.
Built-in procedure {procedure identifier} called in a functional context.
Built_in {procedure identifier} does not accept arguments of the form 'expr:expr' or 'expr:expr:expr'.
Built_in {function identifier} expects {integer} arguments but has been called with {integer}.

5.0 ERRORS
5.1 FATAL DIAGNOSTICS (COMPILE TIME)

CHR built-in function can be applied only to integer arguments.
Cannot duplicate case constant values; the offending case constant is {identifier}.
Cannot pass a field of a packed record as a VAR parameter.
Cannot pass an element of a packed array as a VAR parameter.
Case constant is not compatible with the case index expression.
Case constants do not exhaust selector type and there is no otherwise.
Case constants exceed cardinality of selector type.
Case selector must be of an ordinal type.
Constant expression out of range.
Constant subscript out of range.
EOLN function can only be applied to files of type TEXT.
Elements of a set of integer must be in the range 0..255. {integer} is out of this range.
Expression components must be constants, variables, or function calls. Identifier {identifier} is none of these.
Field name not a field of current record {identifier}.
Field qualification applied to a non-record.
Files or structured types with file components cannot be used in assignment statements or as value parameters.
First argument to procedure {procdure name} must be of type POINTER.
For procedure or function {identifier}, the number of actual parameters, {integer}, does not the number of formal parameters, {integer}.
Formal and actual array parameters have incompatible index types.
Formal and actual array parameters must have the same component type.
FracDigits format specifier must be of type integer.
{identifier} has an unacceptable type; a program parameter must be an integer, a real, a string, or a file.
Identifier {identifier} is not a legal directive. The options are FORWARD and EXTERNAL.
Identifier {identifier} is unknown in the current scope.
{identifier} is a function and can only be called in expressions.
Illegal actual argument for procedure or function formal parameter.
Label {integer} has already been used on line {line number}.
Label {integer} is not accessible to previous GOTO(s) that referenced it.
Label {integer} was not declared in the current block.
Long form of {procedure identifier} can only be used with

5.0 ERRORS
5.1 FATAL DIAGNOSTICS (COMPILE TIME)

pointers to variant records.
Math built-in functions accept only real and integer arguments.
Multiple definition of {identifier}; definition at line {line number} remains in effect.
Non-constant identifier {identifier} used in constant definition.
Non-constant identifier {identifier} used where an ordinal constant is required.
Non-record used in a WITH list.
Non-variable {identifier} used as a FOR control variable.
Non-variable {identifier} used as a variable access.
ODD built-in function can be applied only to integer arguments.
ORD built-in function can be applied only to ordinal arguments.
Only an array or a string can be passed to a conformant array parameter.
Only arrays can be subscripted.
Operator 'IN': the ordinal type of the left operand is not compatible with the base type of the right operand.
Operator {operator symbol} is not defined for the left operand.
Operator {operator symbol} is not defined for the left operand which is of type {identifier}.
Operator {operator symbol} is not defined for the operand.
Operator {operator symbol} is not defined for the right operand.
Operator {operator symbol} is not defined for the right operand which is of type {identifier}.
Operator {operator symbol}: the ordinal operands are not compatible.
Operator {operator symbol}: the pointer operands must be of the same type.
Operator {operator symbol}: the set operands must have compatible base types.
Operator {operator symbol}: the string operands must be of the same length.
Ordinal constants other than those of type integer cannot be signed.
PRED built-in function can be applied only to ordinal arguments.
Packing of the actual and formal parameter do not match.
Parameter number {integer}: actual parameter not of same type as VAR formal parameter.
Parameter number {integer}: an actual parameter cannot be a conformant array for a value formal parameter.
Parameter number {integer}: the parameters of {procedure

5.0 ERRORS
5.1 FATAL DIAGNOSTICS (COMPILE TIME)

identifier} cannot be 'expr:expr' or 'expr:expr:expr'.
Predefined file INPUT was not declared as a program parameter.
Predefined file OUTPUT was not declared as a program parameter.
Predefined file {identifier} was not declared as a program parameter.
Procedure {procedure identifier} cannot be the target of an assignment.
Procedure {procedure identifier} was called in a functional context.
Procedure {procedure identifier} must have at least one argument.
Procedure or function {identifier} resolves a FORWARD directive, but contains a parameter list or a return type. The FORWARD declaration is at line {line number}.
Program parameter {identifier} has not been declared as a variable.
ROUND built-in function can be applied only to real arguments.
Real constant identifier {identifier} used where an ordinal is required.
Real constant used where an ordinal is required.
Return types don't match for actual and formal function parameters.
SQR built-in function can be applied only to real or integer arguments.
SUCC built-in function can be applied only to ordinal arguments.
String argument to {function identifier} must be of length 8.
String constant identifier {identifier} used where an ordinal is required.
String constant used where an ordinal is required.
TRUNC built-in function can be applied only to real arguments.
TYPE {identifier} cannot be used as a value parameter. Value parameters cannot be files or structured types which contain files.
Target array type is not compatible with value type.
Target label {integer} of goto was not declared.
Target ordinal type is not compatible with value type.
Target pointer type is not compatible with value type.
Target real type is not compatible with value type.
Target record type is not assignment compatible with the value's type.
Target set type is not compatible with value type.
Target string length must be equal to value string length.
Target string type not compatible with value type.

5.0 ERRORS
5.1 FATAL DIAGNOSTICS (COMPILE TIME)

The FOR control variable {identifier} is unknown in the current scope.
The FOR control variable {identifier} must be declared in the current block.
The FOR control variable {identifier} must not be the control variable of a containing FOR statement.
The FOR control variable, initial value, and final value must be of compatible ordinal types.
The actual argument for a procedure parameter cannot be a function.
The argument of {function identifier} must be a string.
The array arguments of PACK must have the same component type.
The base type for a set of integers must be within 0..255 .
The base type of a set must be an ordinal type.
The bounds of a subrange must be of some ordinal type.
The case constant {identifier} does not match the case constants in the variant record.
The case constant is not in the range of the selector type.
The case index expression must be of some ordinal type.
The component of a packed conformant array cannot be another conformant array.
The constant declaration of {identifier} contains a signed char, which is illegal.
The constant declaration of {identifier} contains a signed boolean, which is illegal.
The constant declaration of {identifier} contains a signed string, which is illegal.
The dereference operator, '^', can only be applied to pointers and files.
The elements of a set constructor must all be of the same ordinal type.
The elements of a set constructor must of some ordinal type.
The expression in a WHILE must be of type boolean.
The expression in an IF must be of type boolean.
The expression in an UNTIL must be of type boolean.
The first argument of PACK must be an unpacked array.
The functions EOF and EOLN cannot have more than one argument.
The identifier {identifier} has been used as a type, but is unknown is the current scope.
The identifier {identifier} has been used as a type, but is not a type.
The identifier {identifier} has been used in a procedure or function call, but is unknown in the current scope.
The identifier {identifier} has been used in a procedure or function call, but is neither.
The identifier {identifier} is unknown in the current scope.

5.0 ERRORS
5.1 FATAL DIAGNOSTICS (COMPILE TIME)

The integer constant {integer} is too big.
The label {integer} is not accessible from this GOTO statement.
The label {integer} is not in the range 0..9999 .
The label {integer} must appear on a statement in the current block.
The lower and upper bounds of a range must be of the same type.
The maximum number of format specifications is 2 -- found {integer} format specifications.
The operands of {operator symbol} are incompatible. Their types are {identifier} and {identifier}.
The parameter list for actual parameter {integer} does not match the parameter list of the formal parameter.
The procedure PAGE can only be applied to files of type TEXT.
The procedure PAGE cannot have more than one argument.
The procedures READ and WRITE must have at least one non file parameter.
The procedures READLN and WRITELN can only be applied to textfiles.
The real constant {real string} is out of range.
The second argument of PACK must be of an ordinal type that is compatible with the index type of the third argument.
The subscript type is not compatible with the array index type.
The third argument of PACK must be a packed array.
The type of the case constant is not compatible with the type of the selector.
This is not an acceptable type for textfile input.
This is not an acceptable type for textfile output.
TotalWidth format specifier must be of type integer.
TotalWidth:FracDigits format can only be used with type REAL.
Tried to assign to function {function identifier} outside of the function.
Undeclared identifier used as actual parameter.
Variable access required. Expressions are not allowed in this context.

5.2 STANDARDS DIAGNOSTICS (COMPILE TIME)

Standards diagnostics are selectable by control statement option. The will be produced in the same manner as the other compile time diagnostics and will have the following format:

{level} *standard* {column number} text

Diagnostic Texts
{information in brackets is variable}

5.0 ERRORS
5.2 STANDARDS DIAGNOSTICS (COMPILE TIME)

Conformant-array as actual value parameter is non-standard.
Conformant-array parameters are non-standard.
Duplication of declaration section is non-standard.
External or FORTRAN directive is non-standard.
Integer > maxint is non-standard.
Mixed order of declarations is non-standard.
OTHERWISE is non-standard.
Predeclared identifier {identifier} is non-standard.
Treating a conformant-array parameter as a string
is non-standard.
Value declaration part is non-standard.

5.3 FATAL DIAGNOSTICS (RUN TIME)

Diagnostic Numbers and Texts

PA595001 Argument to function Ln must be greater than zero.
PA595002 Argument to function Sqrt must not be negative.
PA595003 Attempt to access component of a variant which is not active.
PA595004 Attempt to alter file variable {filename} when a reference exists.
PA595005 Attempt to dispose pointer when a reference exists.
PA595006 Attempt to divide by zero.
PA595007 Attempted DISPOSE of a pointer variable with an invalid value.
PA595008 Attempted DISPOSE of a NIL pointer.
PA595009 Attempted READ when end of file is TRUE on file {filename}.
PA595010 Attempted READ without RESET on file {filename}.
PA595011 Cannot allocate PDT on stack.
PA595012 Cannot allocate file table.
PA595013 Cannot allocate wsa in REWRITE for file {filename}.
PA595014 Cannot allocate wsa in textfile RESET for file {filename}.
PA595015 Control variable of FOR statement out of range.
PA595016 Data overflow in MLP$INPUT_FLOATING_NUMBER.
PA595017 EOF activated for undefined file {filename}.
PA595018 EOF must be TRUE before PUT on file {filename}.
PA595019 EOF must be TRUE before WRITE on file {filename}.
PA595020 EOF must be true prior to use of PAGE on file

5.0 ERRORS
5.3 FATAL DIAGNOSTICS (RUN TIME)

{filename}.

PA595021 EOF must be true prior to use of WRITE on file {filename}.

PA595022 EOF must be true prior to use of WRITELN on file {filename}.

PA595023 EOLN activated for undefined file {filename}.

PA595024 EOLN activated when EOF is TRUE for file {filename}.

PA595025 End of file encountered on file {filename}.

PA595026 Error in MLP$INPUT_FLOATING_NUMBER.

PA595027 File {filename} is undefined prior to use of GET.

PA595028 File {filename} is undefined prior to use of PAGE.

PA595029 File {filename} is undefined prior to use of PUT.

PA595030 File {filename} is undefined prior to use of READ.

PA595031 File {filename} is undefined prior to use of WRITE.

PA595032 File {filename} is undefined prior to use of WRITELN.

PA595033 File mode must be Generation prior to use of PAGE on file {filename}.

PA595034 File mode must be Generation prior to use of PUT on file {filename}.

PA595035 File mode must be Generation prior to use of WRITE on file {filename}.

PA595036 File mode must be Generation prior to use of WRITELN on file {filename}.

PA595037 File mode must be Inspection prior to use of GET on file {filename}.

PA595038 File mode must be Inspection prior to use of READ on file {filename}.

PA595039 Fractional Digits must be greater than or equal to 1.

PA595040 Free of unallocated block in DISPOSE.

PA595041 GET attempted on file {filename} when EOF is TRUE.

PA595042 Identified pointer variable has a NIL value.

PA595043 Identified pointer variable is undefined.

PA595044 Input attempted after end of file {filename} using GET.

PA595045 Input attempted after end of file {filename} using READ.

PA595046 Insufficient space in wsa to perform textfile output on file {filename}.

PA595047 Integer exceeds MAXINT.

5.0 ERRORS
5.3 FATAL DIAGNOSTICS (RUN TIME)

PA595048 Integer math status error.
PA595049 Interactive file {filename} must be a textfile.
PA595050 Invalid argument to CHR -- character does not exist.
PA595051 Invalid argument to PRED -- value does not exist.
PA595052 Invalid argument to SUCC -- value does not exist.
PA595053 Invalid number.
PA595054 Invalid real number.
PA595055 Line length must be <= 150 for interactive input.
PA595056 Lower merge error in DISPOSE.
PA595057 More than 65535 files used.
PA595058 Ordinal value is out of range.
PA595059 Pascal file {filename} must have file organization of SEQUENTIAL.
PA595060 Pascal file {filename} must have record type of VARIABLE.
PA595061 READ attempted when EOF is TRUE on file {filename}.
PA595062 RESET attempted on undefined file {filename}.
PA595063 Record variants in DISPOSE do not match record variants allocated in NEW.
PA595064 Second argument to MOD cannot be zero or negative.
PA595065 Set types not assignment compatible.
PA595066 This is not a valid integer.
PA595067 This is not a valid signed number.
PA595068 Total width is less than 1.
PA595069 Unable to allocate space for NEW variable in Heap.
PA595070 Upper merge error in DISPOSE.
PA595071 Mod by negative modulo.
PA595072 Index expression out of bounds.
PA595073 Value to be assigned is out of bounds.
PA595074 Element expression out of range.
PA595075 Field width must be greater than zero.

5.4 ERRORS NOT DETECTED

In conformance with the ISO and ANSI standards, a list of errors (in the sense of the standards) which are not detected follows:

Reference and access to component of inactive variant.
Pointer-variable of identified-variable denotes nil-value.
Pointer-variable of identified-variable is undefined.
Removal of identifying-value from pointer-type of identified

variable while reference exists.
Alteration of file-variable f while reference to f^ exists.
Set expression as value parameter which has elements not included in the base type of the formal parameter.
Buffer-variable undefined immediately prior to use of PUT.
Buffer-variable value not assignment compatible with variable access for READ.
Expression value not assignment compatible with buffer variable for WRITE.
Different variant specified for active variant created by NEW(p,c1,..,cn) .
DISPOSE(p) used when identifying-value created by NEW(p,c1,..,cn) .
DISPOSE(p,k1,..,km) applied to variable created by NEW different number of variants.
DISPOSE(p,k1,..,km) applied to variable created by NEW different variant list.
Undefined parameter of a pointer-type to DISPOSE.
Use of variable created by NEW(p,c1,..,cn) in assignment statement or actual parameter.
Value returned by TRUNC(x) not in range -MAXINT..MAXINT.
Value returned by ROUND(x) not in range -MAXINT..MAXINT.
f undefined when EOF(f) activated.
f undefined when EOLN(f) activated.
Use of undefined variable, or portion thereof.
Undefined function result upon completion of function.
Set expression in assignment statement which has elements not included in the base type of the variable-access.
No case constant corresponding to the value of the case-index of a CASE statement.

Table of Contents