

Date

February 28, 1989

Form—PCN number

1169851-001

Title

A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation (Relative to the Mark 3.7.2 System Software

Description

Release)

This letter announces the availability of PCN-001 to the A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation, form number 1169851, dated July 1987. This PCN contains revisions and additions relative to the Mark 3.7.2 System Software Release.

Changes to the text are indicated by vertical black lines on the affected pages.

Replace These Pages

vii	431
ix through xvii	455
69 through 77	543 through 545
115	551 through 567
119	
125	
133	
191	
229	
251	
255 through 257	
287 through 289	
313	
385	
395	
417	

Add These Pages

134A
396A
398A
404A

Retain the PCN cover sheet as a record of changes made to the basic publication.

Copyright © 1989 Unisys Corporation
Unisys Corporation
All Rights Reserved

Page Status

Page	Issue
iii through vi	Original
vii through xviii	PCN-001
1 through 68	Original
69 through 78	PCN-001
79 through 114	Original
115 through 116	PCN-001
117 through 118	Original
119 through 120	PCN-001
121 through 124	Original
125 through 126	PCN-001
127 through 132	Original
133 through 134B	PCN-001
135 through 190	Original
191 through 192	PCN-001
193 through 228	Original
229 through 230	PCN-001
231 through 250	Original
251 through 252	PCN-001
253 through 254	Original
255 through 258	PCN-001
259 through 286	Original
287 through 290	PCN-001
291 through 312	Original
313 through 314	PCN-001
315 through 384	Original
385 through 386	PCN-001
387 through 394	Original
395 through 396B	PCN-001
397 through 398	Original
398A through 398B	PCN-001
399 through 404	Original
404A through 404B	PCN-001
405 through 416	Original
417 through 418	PCN-001
419 through 430	Original
431 through 432	PCN-001
433 through 454	Original
455 through 456	PCN-001
457 through 542	Original
543 through 568	PCN-001

E COMPARISON WITH ANSI PASCAL

Unisys Pascal and ANSI Pascal are compared in the areas of implementation-defined features, implementation-dependent features, and Unisys extensions to the ANSI standard.

F ERROR DETECTION AND REPORTING

Compile-time and run-time detection of errors is described. The errors defined in ANSI Pascal that are not detected in Unisys Pascal are also described.

G SYSTEM-SPECIFIC INFORMATION

All information applicable only to the implementation of Pascal on the A Series systems is provided.

H COMPILING PASCAL MODULES

A description for structuring a source file using modules to build a program and to build a library is given.

In addition, an explanation of railroad diagrams, a glossary, a bibliography, and an index appear at the end of this manual.

RELATED PRODUCT INFORMATION

This list contains the Unisys documents you should read to best understand the material in this manual.

<u>Title</u>	<u>Form No.</u>
BNA Architectural Description Reference Manual, Volume 1	1132172
BNA Host Services Operations Guide	1170339
I/O Subsystem Programming Reference Manual	1169984
Operator Display Terminal (ODT) System Commands Operations Reference Manual	1169612
Pascal Programming Reference Manual, Volume 2: Product Interfaces	1170107
System Software Utilities Operations Reference Manual	1170024

COMPLIANCE STATEMENT

The Unisys Pascal compiler for A Series systems ("Unisys Pascal") complies with the requirements of ANSI/IEEE 770X3.97-1983 and with Level 0 ISO 7185 with the exception that the following features are not currently implemented:

1. The declaration of files and textfiles as components of structured types
2. Dynamic variables of type file or type textfile

Further information on the relationship of Unisys Pascal to ANSI Pascal is contained in the "Comparison with ANSI Pascal" and the "Error Detection and Reporting" appendixes.

See also

Comparison with ANSI Pascal	509
Error Detection and Reporting	517

CONTENTS

ABOUT THIS MANUAL	111
1 PROGRAM STRUCTURE.	1
1.1 PROGRAMS	2
PROGRAM PARAMETERS	4
BLOCKS	7
Scope.	7
Activations.	11
1.2 LIBRARIES.	13
USAGE CLAUSE	16
Sharing Option	16
Duration Option.	18
Matching Option.	19
INTERFACE PART	20
Interface Part for Libraries Using Modules	21
Referencing a Library Entry Point.	21
Parameter Type Matching.	23
1.3 MODULES.	33
2 DECLARATIONS AND DEFINITIONS	39
2.1 EXPORT DECLARATION	41
2.2 IMPORT DECLARATION	44
2.3 LABEL DECLARATIONS	46
2.4 CONSTANT DEFINITIONS	47
2.4.1 SIMPLE CONSTANT DEFINITION	48
2.4.2 STRUCTURED CONSTANT DEFINITION	52
ARRAY VALUE.	54
RECORD VALUE	56
SET CONSTANT EXPRESSION.	59
VLSTRING CONSTANT EXPRESSION	61
COMPONENT CONSTANT	62
INDEX CONSTANT	63
FIELD-DESIGNATED CONSTANT.	64
STRUCTURED CONSTANT EXAMPLE.	65
2.5 TYPE AND SCHEMA DEFINITIONS.	68
2.5.1 TYPE AND SCHEMA CONCEPTS	71
TYPES.	71
ORDINAL TYPES.	73
EXPLICIT TYPES	74
TYPE IDENTIFIERS	76
ARRAY SCHEMATA	77
2.5.2 TYPE COMPATIBILITY	78
SAME TYPES	78
COMPATIBLE TYPES	80
ASSIGNMENT COMPATIBILITY	82
2.5.3 TYPE DESCRIPTIONS.	85
ARRAY TYPES.	85
STRING TYPES	89
BIT TYPES.	90

	BOOLEAN TYPES.	91
	CHAR TYPES	92
	ENUMERATED TYPES	93
	FILE TYPES	95
	FIXED-POINT TYPES.	97
	HEX TYPES.	100
	INTEGER TYPES.	101
	POINTER TYPES.	102
	REAL TYPES	103
	RECORD TYPES	104
	SET TYPES.	111
	SUBRANGE TYPES	113
	TEMPLATE TYPES	115
	TEXTFILE TYPES	117
	VARIABLE-LENGTH STRING (VLSTRING) TYPES.	118
	VERSIONOPTION TYPES.	119
	X-BOOL TYPES	120
	X-NUM TYPES.	122
	Binary	122
	Digits	123
	S_digits	124
	Digits_s	125
	U_display.	126
	Z_display.	127
	Display_z.	128
	S_display.	129
	Display_s.	130
	X-REAL TYPES	131
	X-WORD TYPES	132
	Word48(s).	133
	Word96(s).	133
	X-INTEGERS TYPES.	134
	Integer48.	134
	Integer96.	134A
2.5.4	ARRAY SCHEMA DEFINITIONS	135
2.6	VARIABLE DECLARATIONS.	139
2.7	LIBRARY DECLARATIONS	142
2.8	PROCEDURE AND FUNCTION DECLARATIONS.	144
	PROCEDURE DECLARATION.	145
	FUNCTION DECLARATION	149
	FORMAL PARAMETER LISTS	153
	ACTUAL PARAMETER LISTS AND PARAMETER MATCHING.	156
3	STATEMENTS	159
	ASSIGNMENT STATEMENTS.	161
	CASE STATEMENTS.	162
	COMPOUND STATEMENTS.	165
	FOR STATEMENTS	166
	GOTO STATEMENTS.	169
	IF STATEMENTS.	172
	PROCEDURE INVOCATION STATEMENTS.	174
	REPEAT STATEMENTS.	176
	WHILE STATEMENTS	177

	WITH STATEMENTS.	178
4	EXPRESSIONS.	181
4.1	EXPRESSION CONCEPTS.	183
	ARITHMETIC EXPRESSIONS	183
	ORDINAL EXPRESSIONS.	183
	PRECEDENCE OF OPERATORS.	184
	FUNCTION DESIGNATORS	186
	SCHEMA DISCRIMINANT.	188
4.2	EXPRESSIONS BY TYPE.	189
	BIT PATTERN EXPRESSIONS.	189
	BOOLEAN EXPRESSIONS.	192
	CHAR EXPRESSIONS	195
	ENUMERATED EXPRESSIONS	197
	FIXED-POINT EXPRESSIONS.	199
	INTEGER EXPRESSIONS.	205
	POINTER EXPRESSIONS.	208
	REAL EXPRESSIONS	210
	RELATIONAL EXPRESSIONS	213
	Arithmetic Relation.	214
	Bit Pattern Relation	214
	Ordinal Relation	215
	Pointer Relation	216
	Set Relation	217
	String Relation.	218
	SET EXPRESSIONS.	219
	STRING EXPRESSIONS	221
	VLSTRING EXPRESSIONS	222
	X-BOOL EXPRESSIONS	224
	X-NUM EXPRESSIONS.	225
	X-REAL EXPRESSIONS	226
	X-WORD EXPRESSIONS	227
5	PREDEFINED PROCEDURES AND FUNCTIONS.	229
5.1	FILE HANDLING PROCEDURES AND FUNCTIONS	230
5.1.1	I/O CONCEPTS	232
	TERMINOLOGY.	233
	Standard Files and Textfiles	233
	Inspection Mode and Generation Mode.	233
	Buffer Variables	233
	File Attributes.	234
	Logical and Physical Files	234
	Permanent and Temporary Files.	235
	STANDARD FILES	237
	Inspection Mode Operations on Standard Files	237
	Generation Mode Operations on Standard Files	239
	Inspection/Generation Operations on Standard Files	241
	Representation of Standard Files	242
	TEXTFILES.	243
	Inspection Mode Operations on Textfiles.	243
	Generation Mode Operations on Textfiles.	246
	Representation of Textfiles.	247
	Textfiles "Input" and "Output"	249

	USE OF FILE ATTRIBUTES	250
	I/O EXCEPTION HANDLING	253
	READING FROM AND WRITING TO REMOTE FILES	259
	Textfiles.	259
	Fixed-Length String Files.	260
	Variable-Length I/O.	261
	USING THE EOF FUNCTION AND REMOTE TEXTFILES.	265
5.1.2	PROCEDURE AND FUNCTION DESCRIPTIONS.	268
	ADDSTATION PROCEDURE	268
	CLOSE PROCEDURE.	269
	DELETSTATION PROCEDURE.	272
	DFHVALUE FUNCTION.	273
	EOF FUNCTION	274
	EOLN FUNCTION.	275
	FILEVALUE FUNCTION	276
	GET PROCEDURE.	277
	GET_BYTES PROCEDURE.	279
	HAPPENED FUNCTION.	281
	IORES FUNCTION	282
	OPEN PROCEDURE	283
	PAGE PROCEDURE	287
	PUT PROCEDURE.	288
	PUT_BYTES PROCEDURE.	290
	READ PROCEDURE	294
	READ TEXTFILE PROCEDURE.	295
	READLN PROCEDURE	302
	RESET PROCEDURE.	303
	REWRITE PROCEDURE.	304
	SEEK PROCEDURE	305
	SKIPTOCHANNEL PROCEDURE.	306
	UPDATE PROCEDURE	307
	WRITE PROCEDURE.	308
	WRITE TEXTFILE PROCEDURE	309
	Boolean Expression	310
	Char Expression.	311
	Integer Expression	311
	Vlstring Expression.	312
	Fixed-Point Expression	312
	Real Expression.	313
	WRITELN PROCEDURE.	315
5.2	TYPE TRANSFER PROCEDURES AND FUNCTIONS	316
	BIT PATTERN TYPE TRANSFER FUNCTION	318
	BOOLEAN TYPE TRANSFER FUNCTION	319
	CHR FUNCTION	320
	FIXED-POINT TYPE TRANSFER FUNCTION	321
	INTEGER TYPE TRANSFER FUNCTION	323
	ORD FUNCTION	324
	ORDINAL TYPE TRANSFER FUNCTION	325
	PACK PROCEDURE	326
	REAL TYPE TRANSFER FUNCTION.	328
	UNPACK PROCEDURE	329
	X-BOOL TYPE TRANSFER FUNCTION.	331
	X-NUM TYPE TRANSFER FUNCTION	332

	X-REAL TYPE TRANSFER FUNCTION.	334
	X-WORD TYPE TRANSFER FUNCTION.	335
5.3	DYNAMIC ALLOCATION PROCEDURES.	337
	DISPOSE PROCEDURE.	341
	MARK PROCEDURE	342
	NEW PROCEDURE.	343
	RELEASE PROCEDURE.	344
5.4	LIBRARY HANDLING PROCEDURES AND FUNCTIONS.	345
	CANCEL PROCEDURE	347
	DELINKLIBRARY PROCEDURE.	348
	FREEZE PROCEDURE	349
	LIBRARYVALUE FUNCTION.	350
	LINKLIBRARY PROCEDURE.	351
	LIBRES FUNCTION.	354
5.5	STRING HANDLING PROCEDURES AND FUNCTIONS	356
	CONCAT FUNCTION.	357
	DELETE PROCEDURE	358
	FILLCHAR PROCEDURE	359
	INSERT PROCEDURE	360
	LENGTH FUNCTION.	361
	POS FUNCTION	362
	SCAN FUNCTION.	363
	STRING FUNCTION.	365
5.6	TEMPLATE PROCEDURES.	366
	ADJUST_TEMPLATE PROCEDURE.	367
	ATTACH_TEMPLATE PROCEDURE.	368
5.7	ARITHMETIC FUNCTIONS	370
	ABS FUNCTION	371
	ARCCOS FUNCTION.	371
	ARCSIN FUNCTION.	372
	ARCTAN FUNCTION.	372
	ARCTANH FUNCTION	372
	COS FUNCTION	373
	COSH FUNCTION.	373
	COTAN FUNCTION	373
	ERF FUNCTION	374
	EXP FUNCTION	374
	FIXEDDIV FUNCTION.	374
	GAMMA FUNCTION	375
	LN FUNCTION.	376
	LNGAMMA FUNCTION	376
	LOG FUNCTION	376
	MAX FUNCTION	377
	MIN FUNCTION	379
	RANDOM FUNCTION.	381
	ROUND FUNCTION	382
	SIN FUNCTION	383
	SINH FUNCTION.	383
	SQR FUNCTION	383
	SQRT FUNCTION.	384
	TAN FUNCTION	384
	TANH FUNCTION.	384
	TRUNC FUNCTION	385

5.8	GENERAL PROCEDURES AND FUNCTIONS	386
	ABORT PROCEDURE	387
	ACCEPT PROCEDURE	388
	DATE PROCEDURE	389
	DISPLAY PROCEDURE	391
	ELAPSED TIME FUNCTION	392
	GETATTRIBUTE PROCEDURE	393
	IOTIME FUNCTION	395
	LINENUMBER FUNCTION	396
	MOVE_BYTES PROCEDURE	396A
	ODD FUNCTION	397
	PRED FUNCTION	398
	PROGRAMDUMP PROCEDURE	398A
	RUNTIME FUNCTION	399
	SETATTRIBUTE PROCEDURE	400
	SUCC FUNCTION	402
	TIME PROCEDURE	403
	VERSIONOPTION FUNCTION	404A
	WAIT PROCEDURE	405
6	VARIABLES	407
6.1	VARIABLES BY ACCESS	407
	ENTIRE VARIABLES	408
	INDEXED VARIABLES	409
	FIELD DESIGNATORS	410
	DYNAMIC VARIABLES	411
	BUFFER VARIABLES	413
6.2	VARIABLES BY TYPE	414
	ARRAY VARIABLE	414
	BIT PATTERN VARIABLE	414
	BOOLEAN VARIABLE	414
	CHAR VARIABLE	414
	ENUMERATED VARIABLE	414
	EXPLICIT ARRAY VARIABLE	414
	EXPLICIT RECORD VARIABLE	415
	FILE VARIABLE	415
	FIXED-POINT VARIABLE	415
	INTEGER VARIABLE	415
	POINTER VARIABLE	415
	REAL VARIABLE	415
	RECORD VARIABLE	415
	SET VARIABLE	416
	STRING VARIABLE	416
	STATION VARIABLE	416
	SUBFILE VARIABLE	417
	TEMPLATE VARIABLE	417
	TEXTFILE VARIABLE	417
	VERSIONOPTION VARIABLE	418
	VLSTRING VARIABLE	418
	X-BOOL VARIABLE	418
	X-INTEGGER VARIABLE	418
	X-NUM VARIABLE	418
	X-REAL VARIABLE	418

	X-WORD VARIABLE.	418
6.3	UNDEFINED VARIABLES.	419
7	BASIC COMPONENTS	421
	CHARACTERS AND CHARACTER STRINGS	422
	IDENTIFIERS.	423
	NUMBERS.	425
	FILE ATTRIBUTES AND MNEMONIC VALUES.	427
8	INTERPRETATION OF PROGRAM TEXT	429
	PROGRAM TEXT	430
	TOKEN.	430
	Reserved Word.	431
	Predefined Identifier.	432
	Context-Sensitive Identifier	433
	Special Token.	436
	TOKEN SEPARATOR.	437
	Blank.	437
	Comment.	437
	Record Boundary.	438
A	COMPILER CONTROL RECORDS	439
A.1	OPTION PHRASE.	441
A.2	OPTION EXPRESSIONS	442
A.3	OPTIONS.	443
	ANSI OPTION.	445
	CLEAR OPTION	445
	CODE OPTION.	446
	DELETE OPTION.	446
	ERRORLIMIT OPTION.	447
	ERRORLIST OPTION	447
	HEAPSIZE OPTION.	448
	INCLNEW OPTION	448
	INCLUDE OPTION	449
	IPCCAPABLE OPTION.	451
	LINEINFO OPTION.	451
	LINKAGE OPTION	452
	LIST OPTION.	452
	LISTDOLLAR OPTION.	453
	LISTIMPORT OPTION.	453
	LISTINCL OPTION.	453
	LISTOMITTED OPTION	454
	MAP OPTION	454
	MERGE OPTION	455
	NEW OPTION	455
	NOBOUNDS OPTION.	456
	NOXREFLIST OPTION.	457
	OMIT OPTION.	457
	PAGE OPTION.	458
	SEQUENCE (SEQ) OPTION.	458
	SEQUENCE BASE OPTION	459
	SEQUENCE INCREMENT OPTION.	460
	STATISTICS OPTION.	460

	STRINGS OPTION	461
	TARGET OPTION.	462
	TTYWRITE OPTION.	463
	USER OPTIONS	465
	USERDEBUG OPTION	465
	VERSION OPTION	466
	VOID OPTION.	469
	WARNSUPR OPTION.	470
	XREF OPTION.	470
	XREFFILES OPTION	471
	XREFLATER OPTION	472
B	COMPILER FILES	473
	INTERACTIVE (CANDE) COMPILATION.	477
	BATCH (WFL) COMPILATION.	478
C	DATA REPRESENTATION.	481
C.1	SIMPLE TYPES	483
	BOOLEANS	483
	CHARACTERS	484
	ENUMERATIONS	485
	FIXED-POINTS	485
	INTEGERS	487
	REALS.	487
	SUBRANGES.	488
C.2	STRUCTURED TYPES	489
	ARRAYS	489
	Unpacked Arrays.	489
	Packed Arrays.	490
	FILES.	491
	RECORDS.	491
	Unpacked Records	491
	Packed Records	491
	SETS	494
	Unpacked Sets.	494
	Packed Sets.	494
	TEXTFILES.	494
	VLSTRINGS.	495
C.3	POINTERS	496
C.4	UNDEFINED OPERANDS	496
D	EBCDIC AND ASCII CHARACTER SETS.	497
E	COMPARISON WITH ANSI PASCAL.	509
	IMPLEMENTATION-DEFINED FEATURES.	510
	IMPLEMENTATION-DEPENDENT FEATURES.	513
	EXTENSIONS TO STANDARD PASCAL.	515
F	ERROR DETECTION AND REPORTING.	517
G	SYSTEM-SPECIFIC INFORMATION.	521
H	COMPILING PASCAL MODULES	523

UNDERSTANDING RAILROAD DIAGRAMS 525

GLOSSARY. 535

BIBLIOGRAPHY. 545

INDEX 547

Declarations and Definitions

<type definition>

```

-----<array type identifier>-- = --<array type>-----|
|
|-<bits type identifier>-- = --<bits type>-----|
|
|-<Boolean type identifier>-- = --<Boolean type>-----|
|
|-<char type identifier>-- = --<char type>-----|
|
|-<enumerated type identifier>-- = --<enumerated type>-----|
|
|-<file type identifier>-- = --<file type>-----|
|
|-<fixed-point type identifier>-- = --<fixed-point type>-----|
|
|-<hex type identifier>-- = --<hex type>-----|
|
|-<integer type identifier>-- = --<integer type>-----|
|
|-<pointer type identifier>-- = --<pointer type>-----|
|
|-<real type identifier>-- = --<real type>-----|
|
|-<record type identifier>-- = --<record type>-----|
|
|-<set type identifier>-- = --<set type>-----|
|
|-<subrange type identifier>-- = --<subrange type>-----|
|
|-<template type identifier>-- = --<template type>-----|
|
|-<textfile type identifier>-- = --<textfile type>-----|
|
|-<versionoption type identifier>-- = --<versionoption type>-----|
|
|-<vlstring type identifier>-- = --<vlstring type>-----|
|
|-<x-bool type identifier>-- = --<x-bool type>-----|
|
|-<x-integer type identifier>-- = --<x-integer type>-----|
|
|-<x-num type identifier>-- = --<x-num type>-----|
|
|-<x-real type identifier>-- = --<x-real type>-----|
|
|-<x-word type identifier>-- = --<x-word type>-----|

```

```

<array type identifier>
<bits type identifier>
<Boolean type identifier>
<char type identifier>
<enumerated type identifier>
<file type identifier>
<fixed-point type identifier>
<hex type identifier>
<integer type identifier>
<pointer type identifier>
<real type identifier>
<record type identifier>
<set type identifier>
<subrange type identifier>
<template type identifier>
<textfile type identifier>
<versionoption type identifier>
<vlstring type identifier>
<x-bool type identifier>
<x-integer type identifier>
<x-num type identifier>
<x-real type identifier>
<x-word type identifier>

--<identifier>--|

```

Every variable, constant, and function has an associated "type" that defines its range of valid values, its internal and external representation, and the operations that can be performed on it. The <type and schema definitions> allow user-defined types to be named and their characteristics to be specified.

The details of each type's actual representation are described in the "Data Representation" appendix.

General concepts that apply to types, such as type categories, type compatibility, and type identifiers, are discussed under "Type and Schema Concepts" in this section. Types themselves are listed in alphabetical order under separate headings.

See also

Array Schema Definitions	135
Data Representation	481

2.5.1 TYPE AND SCHEMA CONCEPTSTYPES

<type>

```

-----<simple type>-----|
|
|-<structured type>-----|
|
|-<pointer type>-----|
|
|-<parameterized bit pattern type>-|
|
|-<explicit type>-----|
|
|-<template type>-----|

```

Types can be classified into categories that relate to their structure: simple types, structured types, pointer types, parameterized bit pattern types, explicit types, and template types.

<simple type>

```

-----<Boolean type>-----|
|
|-<char type>-----|
|
|-<enumerated type>--|
|
|-<fixed-point type>-|
|
|-<integer type>-----|
|
|-<real type>-----|
|
|-<subrange type>-----|

```

Variables of simple types have only one component. The predefined types Boolean, char, fixed-point, integer, and real are simple types. User-defined derivatives of these predefined types, as well as enumerated types and subrange types, are also simple types.

<structured type>

```

-----<array type>-----|
|<set type>-----|
|<record type>-----|
|<file type>-----|
|<textfile type>-----|
|<versionoption type>--|
|<vlstring type>-----|

```

Variables of structured types have multiple components, which can be of one or more simple types or can be structured themselves. Arrays, sets, records, files, textfiles, version options, and vlstrings are structured types.

<pointer type>

Variables of pointer types contain values that are references to variables of simple or structured types.

<parameterized bit pattern type>

```

----<bits type>----|
|                    |
|<hex type>--|

```

Variables of parameterized bit pattern types represent bit patterns of a specified length.

<explicit type>

Variables of explicit types represent numeric values in an explicitly specified format.

<template type>

Template types change how variables of explicit array type or explicit record type are viewed or used.

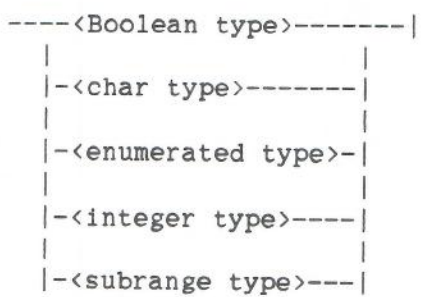
A template can be attached to an <explicit array variable> or an <explicit record variable> at any byte offset (refer to "ATTACH_Template Procedure" in the "Predefined Procedures and Functions" section for additional information). After the template is attached, it acts as if it were a variable of the type for which it is a template. For example, a template attached to an <explicit array variable> acts as an <explicit array variable>. A <template type> cannot be used unless it is attached to an explicit array or record.

See also

ATTACH_Template Procedure 368

ORDINAL TYPES

<ordinal type>



Most simple types are also "ordinal types." In an ordinal type, the values have a well-defined sequential relationship. Each value is assigned an "ordinal number" that uniquely identifies its position in the sequence. Thus, a value of an ordinal type can have a "successor" and a "predecessor" in the sequence. Values can also be compared (for example, greater than or less than) based on their ordinal numbers. The exceptions are <real type>s and <fixed-point type>s. They are simple types but not ordinal types.

EXPLICIT TYPES

<explicit type>

```

----<bit pattern type>-----|
|                               |
| -<explicit array type>--|   |
|                               |
| -<explicit record type>-|   |
|                               |
| -<x-bool type>-----|   |
|                               |
| -<x-num type>-----|   |
|                               |
| -<x-real type>-----|   |
|                               |
| -<x-word type>-----|   |

```

An <explicit type> is a type with the internal data format (and location, if within a record) that is explicitly specified. The <explicit type> is used primarily for importing data types from non-Pascal external interfaces such as the Advanced Data Dictionary System (ADDS).

<bit pattern type>

```

----<bits type>-----|
|                               |
| -<char type>----|   |
|                               |
| -<hex type>----|   |
|                               |
| -<string type>-|   |

```

The <bit pattern type>s are a class of predefined types having associated values that are bit patterns.

<explicit array type>

This type is an <array type> designated as PACKED with a <component type> that is an <explicit type> represented by an integral number of bytes.

Declarations and Definitions

<explicit record type>

This type is a <record type> having <field type>s that are all <explicit type>s, and has specified field locations.

<x-bool type>

The <x-bool type> represents a Boolean value in an explicitly specified format.

<x-num type>

The <x-num type> represents numeric values in an explicitly specified format.

<x-real type>

The <x-real type> defines the set of floating-point values that can be represented by a single-precision operand.

<x-word type>

The <x-word type> defines the explicit single-precision and double-precision numeric types.

TYPE IDENTIFIERS

<type identifier>

```

---- Boolean -----|
|
| - char -----|
|
| - integer -----|
|
| - real -----|
|
| - text -----|
|
| -<array type identifier>-----|
|
| -<bits type identifier>-----|
|
| -<Boolean type identifier>-----|
|
| -<char type identifier>-----|
|
| -<enumerated type identifier>----|
|
| -<file type identifier>-----|
|
| -<fixed-point type identifier>---|
|
| -<hex type identifier>-----|
|
| -<integer type identifier>-----|
|
| -<pointer type identifier>-----|
|
| -<real type identifier>-----|
|
| -<record type identifier>-----|
|
| -<set type identifier>-----|
|
| -<subrange type identifier>-----|
|
| -<template type identifier>-----|
|
| -<textfile type identifier>-----|
|
| -<versionoption type identifier>-|
|
| -<vlstring type identifier>-----|
|
| -<x-bool type identifier>-----|

```


Declarations and Definitions

```

|<x-integer type identifier>-----|
|
|<x-num type identifier>-----|
|
|<x-real type identifier>-----|
|
|<x-word type identifier>-----|

```

In <type and schema definitions> and <variable declarations>, a type can usually be defined in one of two ways:

1. As a new type; that is, by using the <new array type>, <new bits type>, <new enumerated type>, <new file type>, <new hex type>, <new pointer type>, <new record type>, <new set type>, <new subrange type>, or <new vlstring type> syntax
2. As a derived type, where an <identifier> that has already been defined (or was predefined) as a type identifier is specified

In other contexts requiring a type specification, new types are not allowed and previously defined <type identifier>s must be used.

ARRAY SCHEMATA

Schemata is the plural of the word schema, which means outline or plan. An array schema definition is a formal syntactical outline describing a collection of array types called a schema family. Each array type in this collection has the same primary characteristics as the others: the same component type, the same number of dimensions, and the same ordinal types for the subscripts of the corresponding dimensions.

Array schemata allow arrays with different bounds and different numbers of elements, but with types belonging to the same schema family, to be passed as actual parameters to the same formal parameter. The type of this formal parameter is designated by a schema identifier denoting the schema family, rather than a type identifier denoting the individual array type.

With array schemata, a formal parameter of a procedure or function can be denoted by an array schema identifier. The procedure or function can then handle any array type that is a member of the schema family.

See also

Array Schema Definitions.	135
Array Types	85

2.5.2 TYPE COMPATIBILITYSAME TYPES

Because types can be defined in different ways, it is not always clear when two types are actually the same type. The concept of "same type" is used when describing how <variable parameter>s are matched in procedure and function invocations. More important, the definition of "same type" is used to define compatible types and assignment compatibility, which are described under later headings.

The <type identifier>s T1 and T2 are the same type if one of the following conditions is true:

1. One type is defined to be equal to the other.
2. Both types are the same type as a third type.

The simplest case of "same type" is when T1 is defined as equal to T2, as shown in the example below:

```
TYPE T1 = T2; { same type, by condition 1 }
```

Condition 2 describes the case in which T1 and T2 have a common ancestor, the simplest case of which is the following:

```
TYPE T3 = INTEGER;
   T1 = T3; { same type, by condition 1 }
   T2 = T3; { same type, by condition 1 }
{ T1 is the same type as T2, by condition 2 }
```

In the example above, T1 is the same type as T3, by condition 1. Similarly, T2 is also the same type as T3. Thus, T1 and T2 are the same type because they are both the same type as T3. By a somewhat longer chain, T1 and T2 can be recognized as the same type in the following example:

```
TYPE T5 = INTEGER;
   T4 = T5;
   T3 = INTEGER;
   T2 = T4;
   T1 = T3;
```

Declarations and Definitions

TEMPLATE TYPES

<template type>

```

---- template ----- ( --<template domain type>-- ) --|
|                                     |
|- const_template -|

```

<template domain type>

```

----<explicit array type identifier>-----|
|                                     |
|-<explicit record type identifier>-|

```

A <template type> is a pattern that can be attached to an explicitly represented array or an explicitly represented record at any byte offset. Refer to "Template Procedures" for additional information.

The <template type>s are a Unisys extension to ANSI Pascal.

See also

Template Procedures 366

Example

```

type binary_length_type = binary(16);
   text_area = packed array [1..65535] of char;

msg = record
   length(1) : binary_length_type;
   message(3) : text_area;
end;
msg_template = template(msg);

```

A variation of the "template" construct has been implemented that allows a template to be attached to a read-only parameter. The new construct is a read-only template, spelled "const_template".

Example

```

program demo_const_template;
type
  rec =
    record
      first(1): packed array [1..1] of char;
      rest    : packed array [1..10] of char;
    end;
  readonly_template = const_template( rec );
  large_area = packed array[1..65535] of char;
var
  data_area: large_area;
procedure takes_var_param( var r: rec );
begin ... end;
procedure p( const area: large_area );
  var t: readonly_template;
      c: char;
  begin
    attach_template( t, area, 1 );
    c := t.first;
    { The following statement results in a syntax error because }
    { a read-only template variable cannot be assigned to:      }
    t.first := c;
    { The following statement results in a syntax error because }
    { a read-only template variable cannot be passed as a      }
    { "var" parameter:                                          }
    takes_var_param( t );
  end;
begin
  ...
  p( data_area );
end.

```

A <template variable> declared with the "const_template" form of <template type> is a read-only <template variable>.

A read-only <template variable> is similar to a <template variable> declared with the "template" form of <template type>, except that it can be attached (using the "attach_template" procedure) to an <explicit array variable> or <explicit record variable> that is a read-only parameter. Also, the read-only <template variable> cannot be the target of an assignment statement or be used as the actual parameter corresponding to a formal "var" parameter. That is, the same restrictions that apply to a read-only parameter also apply to a read-only <template variable>.

Example

```

type str1 = string(10);
   str2 = string(80);
   str3 = str2;

```

Type "str1" is a <vlstring type identifier> that has a <maximum length> of 10 characters. "str2" and "str3" are compatible <vlstring type identifier>s that have a <maximum length> of 80 characters.

VERSIONOPTION TYPES

```

<versionoption type>
    --<versionoption type identifier>--|

<release>
<cycle>
<patch>

    -- <integer variable> --|

```

The <versionoption type> is a record type containing three integer fields. The <versionoption type> is used with the <versionoption function> to obtain the version assigned using the VERSION compiler control option. The <versionoption type> fields are in the following ranges:

Parameter	Range
<release>	0..99
<cycle>	0..999
<patch>	0..9999

The fields of a <versionoption type> variable can be accessed the way fields of any record can be accessed, using dot qualification or the WITH statement. For an example of how to use the <versionoption type> and the <versionoption function>, refer to the VERSIONOPTION FUNCTION in the "Predefined Procedures and Functions" section.

See also
 VERSIONOPTION FUNCTION. 404A

X-BOOL TYPES

<x-bool type>

```

---- Boolean4 ----|
|                   |
|- Boolean1 -|

```

An <x-bool type> represents a Boolean value in an explicitly specified format, but cannot be treated directly as a Boolean value within a program. The explicit Boolean type must first be converted to a Boolean type through a type transfer process using the <x-bool type transfer function> before it can be referenced as a Boolean value. An explicit Boolean type can be transferred to either a <Boolean type> or a <bit pattern type>.

Attempting to transfer a value of a <bit pattern type> into an <x-bool type> that will not hold the value produces a range error, and vice versa.

Also, comparisons between explicit Boolean values cannot be performed. The <x-bool type> values must be converted to either <Boolean type> or <bit pattern type> values to be compared.

Like the <x-word type>, <x-bool type>s are used mainly to import COBOL-formatted records from the Advanced Data Dictionary System (ADDS).

The <x-bool type> is a Unisys extension to ANSI Pascal.

Boolean4

A variable of type Boolean4 requires exactly 4 bits to represent a Boolean value in a packed 4-bit format. Fields of type Boolean4 are constrained to start on 4-bit boundaries within a record.

Example

```

var bool4_var1, bool4_var2 : Boolean4;
begin
  bool4_var1 := Boolean4(4'1');
  bool4_var2 := Boolean4(true);
end;

```

Declarations and Definitions

Examples

The example below shows how a variable of type `s_digits(n)` can be used. Given the definition for the `s_digits` type variable, the resulting relationships are true.

```
type s_digits8 = s_digits(8);

integer(s_digits8 (4'C1234567')) = 1234567
integer(s_digits8 (4'D1234567')) = -1234567
```

The example below shows how a variable of type `s_digits(n,s)` can be used. Given the definitions for the `s_digits` type and `sfixed` type variables, the resulting relationship is true.

```
type s_digits5_8 = s_digits(5,8);
   sfixed4_8     = sfixed(4,8);

sfixed4_8 (s_digits5_8 (4'D1234')) = - 0.00001234
```

Digits s

A variable of type `digits_s(n)` represents an integer value in trailing sign, packed decimal format. In all other respects, `digits_s(n)` is the same as `s_digits(n)`.

When the optional `<scale factor>` is used, a variable of type `digits_s(n,s)` requires exactly $n*4$ bits to represent a fixed-point value in the range $-((10^{(n-1)}-1)/(10^s))$ to $((10^{(n-1)}-1)/(10^s)$.

Values for variables of type `digits_s` can be up to 24 digits long (1 digit is for the sign). Fields of `s_digits` type are constrained to start on 4-bit boundaries within records.

Examples

The example below shows how a variable of type `digits_s(n)` can be used. Given the definition for the `digits_s` type variable, the resulting relationships are true.

```
type digits_s8 = digits_s(8);
```

```
integer(digits_s8 (4'1234567C')) = 1234567
integer(digits_s8 (4'1234567D')) = -1234567
```

The example below shows how a variable of type `digits_s(n,s)` can be used. Given the definition for the `digits_s` type and `sfixed` type variables, the resulting relationships are true.

```
type digits_s6_m3 = digits_s(6,-3);
   sfixed5_m3     = sfixed(5,-3);

sfixed5_m3 (digits_s6_m3 (4'12345C')) = 12345000
sfixed5_m3 (digits_s6_m3 (4'12345D')) = -12345000
```

U display

A variable of type `u_display(n)` represents an integer value in unsigned, unpacked decimal format. It requires exactly "n" bytes and represents an integer in the range 0 (zero) to $(10^{**n})-1$. The value of "n" must be between 1 and 23, inclusive.

When the optional <scale factor> is used, a variable of type `u_display(n,s)` requires exactly "n" bytes to represent a fixed-point value in the range 0 (zero) to $((10^{**n})-1)/(10^{**s})$ in an unsigned, display numeric format.

Values for variables of type `u_display` can be up to 23 bytes long.

Examples

The example below shows how a variable of type `u_display(n)` can be used. Given the definition for the `u_display` type variable, the resulting relationship is true.

```
type u_display4 = u_display(4);

integer(u_display4 ('1234')) = 1234
```


Declarations and Definitions

Word48(s)

A variable of type word48(s) requires exactly 48 bits to represent a fixed-point value in single-precision format. The <scale factor>, "s", must be between 47 and -68, inclusive. Fields of type word48 are constrained to start on byte boundaries within a record.

Example

```

type word48_2      = word48(2);
   word48_m2      = word48(-2);
   fixed3_2       = fixed(3,2);
   fixed10_m2     = fixed(10,-2);

```

Given the preceding definitions, the following relationships are true:

```

fixed3_2 (word48_2 (4'0000000000FF')) = 2.55
fixed10_m2 (word48_m2 (4'0000000000FF')) = 25500

```

Word96(s)

A variable of type word96(s) requires exactly 96 bits to represent a fixed-point value in double-precision format. The <scale factor>, "s", must be between 47 and -68, inclusive. Fields of type word96 are constrained to start on byte boundaries within a record.

Example

```

type word96_2      = word96(2);
   word96_m2      = word96(-2);
   sfixed10_4     = fixed(10,4);
   fixed4_m2      = fixed(4,-2);

```

Given the above definitions, the following relationships are true:

```

sfixed10_4 (word96_2 (4'4000000000FF000000000000')) = -2.55
fixed4_m2 (word96_m2 (4'0000000000FF000000000000')) = 25500

```

X-INTEGER TYPES

<x-integer type>

```

---- integer48 ----|
|                   |
|- integer96 -|

```

Integer48

For convenience, integer48 is included as a predefined word48(0) type that represents a single-precision integer value. Thus, type transfer is possible between integer48 and integer types. All rules concerning type word48 remain the same. Fields of type integer48 are constrained to start on byte boundaries within a record.

Example

```

var int48_var1, int48_var2 : integer48;
begin
  int48_var1 := integer48(4'000000000FF');
  int48_var2 := integer48(4'400000000FF');
end;

```

Given the preceding assignments of int48_var1 and int48_var2, the following relationships are true:

```

integer (int48_var1) = 255
integer (int48_var2) = -255

```

Integer96

Similarly, integer96 is included as a predefined word96(0) type that represents a double-precision integer value. Thus, type transfer is possible between integer96 and integer types. All rules concerning type word96 remain the same. Fields of type integer96 are constrained to start on byte boundaries within a record.

Example

```
var int96_var1, int96_var2 : integer96;
begin
  int96_var1 := integer96 (4'4000000000FF000000000000');
  int96_var2 := integer96 (255);
end;
```

Given the above assignments of int96_var1 and int96_var2, the following relationships are true:

```
integer (int96_var1) = -255
integer (int96_var2) = 255
```


Expressions

The <bits constant>s and <hex constant>s are Unisys extensions to ANSI Pascal.

See also

Type Transfer Procedures and Functions. 316
Variables 407

Example

```
program bit_patterns;

const bitmask = 1'00010001';

var hex_byte : hex(2);
    bits_byte: bits(8);

begin
bits_byte := bits_mask;
hex_byte := bits_byte or 1'10000000';
end.
```

BOOLEAN EXPRESSIONS

<Boolean expression>

```

|<-----<Boolean operator>-----|
|
|-----<Boolean primary>-----|
|
|   NOT   |
|- NOT -|

```

<Boolean operator>

```

---- AND ----|
|
|- CAND -|
|
|- OR ---|
|
|- COR --|

```

<Boolean primary>

```

---- ( --<Boolean expression>-- ) ----|
|
|<Boolean constant>-----|
|<Boolean variable>-----|
|<function designator>-----|
|<relational expression>-----|
|<schema discriminant>-----|
|<component constant>-----|

```

<Boolean constant>

```

---- true -----|
|
|- false -----|
|
|<Boolean constant identifier>-|

```

A <Boolean expression> generates a value of the <Boolean type>.

5 PREDEFINED PROCEDURES AND FUNCTIONS

<predefined procedure>

```

----<file handling procedure>-----|
|                                     |
| -<type transfer procedure>-----|
| -<dynamic allocation procedure>-|
| -<library handling procedure>---|
| -<string handling procedure>----|
| -<template handling procedure>--|
| -<general procedure>-----|

```

<predefined function>

```

----<file handling function>-----|
|                                     |
| -<type transfer function>-----|
| -<library handling function>-|
| -<string handling function>--|
| -<arithmetic function>-----|
| -<general function>-----|

```

Many Pascal features, including input/output (I/O) facilities and dynamic variables, are made available through predefined procedures and functions. Although procedures and functions are syntactically different constructs, this difference is not emphasized in this section. Instead, procedures and functions relating to a particular program application, such as file handling, are grouped together to provide some continuity in their descriptions. Within each group, the procedures and functions are listed alphabetically.

5.1 FILE HANDLING PROCEDURES AND FUNCTIONS

<file handling procedure>

```

-----<addstation procedure>-----|
|
| -<close procedure>-----|
|
| -<deletestation procedure>--|
|
| -<dfhvalue function>-----|
|
| -<get procedure>-----|
|
| -<get_bytes procedure>-----|
|
| -<open procedure>-----|
|
| -<page procedure>-----|
|
| -<put procedure>-----|
|
| -<put_bytes procedure>-----|
|
| -<read procedure>-----|
|
| -<read textfile procedure>--|
|
| -<readln procedure>-----|
|
| -<reset procedure>-----|
|
| -<rewrite procedure>-----|
|
| -<seek procedure>-----|
|
| -<skiptochannel procedure>--|
|
| -<update procedure>-----|
|
| -<write procedure>-----|
|
| -<write textfile procedure>--|
|
| -<writeln procedure>-----|

```


Predefined Procedures and Functions

TABLE 5-2. MAPPING OF ATTRIBUTE TYPES WITH PASCAL DATA TYPES

I/O Type	Pascal Type
integer (without mnemonics)	integer
integer (with mnemonics)	integer, for "setattribute" and "getattribute"; mnemonic, otherwise.
Boolean	Boolean
pointer	string, vlstring
real (numeric)	real
real (SERIALNO)	string or vlstring (6 significant characters)
real (bits -- STATE, ATTVALUE, USERINFO)	real
translatetable	[not available]
event	[available through the <wait procedure>]
STATIONLIST	[available through the <addstation procedure> and the <deletestation procedure>]

Pointer-valued attributes are always assumed by the system to have an EBCDIC representation. Thus, if the STRINGS compiler control option has been set to ASCII, which causes the compiler to assume that all string variables are represented in ASCII, it will automatically translate from ASCII to EBCDIC or EBCDIC to ASCII when setting or accessing pointer-valued attributes.

When accessing a pointer-valued attribute, if the value is shorter than the string variable into which it is being stored, the value is blank filled on the right; if the value is being stored into a vlstring variable, it is not blank filled. If the value is longer than the length of the string variable or the maximum length of the vlstring variable into which it is being stored, an error occurs.

NOTE

A period (".") is not returned as part of the value.

If a null pointer-valued attribute is placed into a string variable, the variable will contain all blanks. If a null pointer-valued attribute is placed into a vlstring variable, the variable has a null value and a length of 0.

When setting the SERIALNO attribute, only the first six characters are used. A value less than six characters in length is blank filled on the right to six characters. A null SERIALNO is specified with a string or vlstring variable at least six characters long, in which the first six characters are NUL ("chr(0)").

The SERIALNO attribute, when accessed, returns a value six characters in length. If the SERIALNO is returned in a string variable longer than six characters, the value is blank filled on the right. If the SERIALNO is returned in a vlstring variable, the value is not blank filled. An error occurs if the value is longer than the length of the string variable or the maximum length of the vlstring variable.

Predefined Procedures and Functions

MAXSUBFILES file attribute, or when a subfile index is not specified and the value of MAXSUBFILES is greater than 1.

Iores(Openall)

This value is returned only if the value of the KIND attribute is PORT. It is returned when an attempt is made to open all subfiles and an error occurs on any one of them.

The exceptions described here can occur when an OPEN is attempted on a PORT file:

Iores(unsupportedfunction)

This error occurs if a request is made for an unsupported function.

Iores(Unsupportedprotocol)

This error occurs when an incompatible protocol is detected.

Iores(Protocolerror)

An error in the protocol was detected.

Iores(Lackofresources)

The OPEN procedure cannot be completed due to lack of resources.

Iores(Hostnotinhostgroup)

The OPEN procedure failed because the host specified in YOURHOST is not a member of the specified YOURHOSTGROUP.

Iores(unauthorizedforag)

The current user is not an authorized user of the specified APPLICATIONGROUP.

Iores(Bnaipcunavailable)

This error occurs if the OPEN procedure failed because the PORT Level Manager was not available to the PORT element.

Iores(Badattributesforopen)

This error occurs if the OPEN procedure failed because the file attributes were set incorrectly.

Iores(Unavailablefunction)

This error occurs if a request is made for an unavailable function.

Iores(Unsupportedintmode)

This error occurs if the intmode attribute was set to a value not supported by BNA.

Iores(Networkworkingnotsupported)

This error occurs if networking is not supported.

Iores(Unsupportedparameterrslt)

This error is returned if the SERVICE attribute is set to TCPUSHEDMSGSERVICE and if OFFER is used as an option for the OPEN procedure.

Iores(Localipcnotsupportedrslt)

This error is returned if the SERVICE attribute is set to TCPUSHEDMSGSERVICE and if YOURHOST is not specified, implying a local OPEN procedure. Local ports are not allowed for TCPUSHEDMSGSERVICE.

Iores(Unsupportedtranslationrslt)

This error is returned if the SERVICE attribute is set to TCPUSHEDSERVICE, if the TRANSLATE attribute is set to the value FULLTRANS, and if the value of the INTMODE and EXTMODE attributes of the port file are not equal. This condition implies that the user requested a translation that is not supported by TCPUSHEDMSGSERVICE.

Get, Put, Read, Readln, Write, Writeln

These exceptions can occur when an attempt is made to read or write a file.

Iores(Ok)

The operation was successful.

Iores(Badindex)

This value is returned only if the value of the KIND attribute is PORT. It is returned when a subfile index is specified that has a value less than 0 or greater than the value of the MAXSUBFILES attribute, or when a subfile index is not specified and MAXSUBFILES is greater than 1.

Iores(Dataerr)

This value is returned if a data error occurred, an invalid value was received by a textfile input operation, or a line overflow occurred on textfile output.

Iores(Deleted)

The value of the FILEORGANIZATION attribute is RELATIVE and a deleted or duplicated record was referenced.

Iores(Parity)

A parity error on the associated physical medium was encountered. Note that the I/O subsystem will report this error only after performing several unsuccessful retries.

Predefined Procedures and Functions

Iores(Porterr)

This value is returned only if the value of the KIND attribute is PORT. It is returned when one of the following errors occurs:

1. A broadcast write failed for one or more subfiles.
2. A "put" with the "dontwait" option failed because no buffer was available.
3. A "get" with the "dontwait" option failed because no data was available.

Iores(Eof)

End-of-file or end-of-page was encountered. Note that this is "end-of-file" as defined by the I/O subsystem, which does not correspond exactly to the value of the Pascal <eof function>. Several conditions can cause this exception to occur without causing the <eof function> to return "true". Also, this exception does not always occur in generation mode, whereas the <eof function> will always return "true" in generation mode. An end-of-page exception occurs when the PAGESIZE attribute has been set and the end of a logical page has been encountered on output.

Iores(Brk)

A break on output from the associated physical device was encountered.

Iores(Timeout)

The TIMELIMIT file attribute has been set, and the specified time limit expired before the I/O operation completed.

Iores(Security)

A security violation was attempted.

Close

These exceptions may occur when the I/O subsystem close routines are invoked to close the file:

Iores(Ok)

The operation was successful.

Iores(Unknown)

An error has occurred, but its nature cannot be determined.

Iores(Notopen)

The file or subfile is already closed.

Iores(Datalost)

All data was not successfully delivered to the destination.

Iores(Recordcount)

A record count error occurred.

Iores(Blockcount)

A block count error occurred.

Iores(Badindex)

This value is returned only if the value of the **KIND** attribute is **PORT**. It is returned when a subfile index less than zero or greater than the value of the **MAXSUBFILES** file attribute is specified, and when a subfile index is not specified and the value of **MAXSUBFILES** is greater than 1.

Iores(Closeall)

This value is returned only if the value of the **KIND** attribute is **PORT**. It is returned when an attempt is made to close all open subfiles and an error occurs on at least one of them.

When handling exception conditions, special attention should be paid to the timing of the operations, particularly regarding textfile I/O. (Refer to the discussion of "Textfiles," especially "Lazy I/O" in this section.)

Under certain rare circumstances, it is possible that an "open" operation on a tape file may perform an implicit "close", in which case a "close" result may be returned. Similarly, a "close" operation on a tape file may perform an implicit open.

The ability to return I/O results is a Unisys extension to ANSI Pascal.

See also

Inspection Mode Operations on Textfiles 243
Iores Function. 282

Predefined Procedures and Functions

PAGE PROCEDURE

<page procedure>

```

-- page -----|
      |         |
      |-( --<textfile variable>-- )-|

```

The <page procedure> is identical in action to a call on the <skiptochannel procedure> with the channel number specified as 1. That is, it causes a <writeln procedure> without carriage control, followed by a skip-to-top-of-page action. If the <textfile variable> is omitted, the action applies to the textfile "output".

If the <page procedure> is invoked for a file that is not associated with a printer, the effect is equivalent to invoking the <writeln procedure>.

An error occurs if the file is in inspection mode or if the file is not open prior to the execution of the <page procedure>.

See also

Skiptochannel Procedure	306
Writeln Procedure	315

PUT PROCEDURE

<put procedure>

```

-- put -- ( ---<textfile variable>----- ) --|
          |-----|
          |-<station variable>-----|
          |-----|
          |-<file variable>-----|
          |-----|
          |-<subfile variable>--| | , --<put option>--|

```

<put option>

```

|<----- , -----|
|                       |
-----/1\- dontwait -----|
|                       |
|-/1\- urgent ---|

```

The <put procedure> writes the value of the buffer variable to the file denoted by <textfile variable>, <file variable>, <subfile variable>, or <station variable>. The value of the buffer variable then becomes undefined.

An error occurs if the file is in inspection mode or if the file is not open prior to execution of the <put procedure>. An error also occurs if a <textfile variable> is specified and the <put procedure> causes the line to exceed the length determined by the value of the MAXRECSIZE file attribute.

A <subfile variable> is meaningful only if the KIND of the file is PORT. If a <subfile variable> is used and the associated <subfile index> is 0, a "broadcast write" (in BNA terminology) is performed; that is, the value of the buffer variable is written to each subfile. If a <subfile index> greater than 0 and less than or equal to the value of the MAXSUBFILES file attribute is specified, the data is written only to the specified subfile. The LASTSUBFILE file attribute is updated to the value of <subfile index> if the operation completes without error. An error occurs if an invalid <subfile index> is specified or if the value of MAXSUBFILES is greater than 1 and a <subfile variable> is not used.

Predefined Procedures and Functions

A <station variable> is meaningful only if the KIND of the file is REMOTE. If a <station variable> is used and the associated <station index> is 0, a "broadcast write" is performed. If a <station index> greater than 0 is specified, the data is written only to the specified station. The LASTSUBFILE file attribute will be updated to the value of <station index> when the operation completes. If a <station variable> is not used, but the specified file is a remote file with multiple stations, the write is directed to the station denoted by the current value of the LASTSUBFILE attribute.

The "dontwait" <open option> is meaningful only if the file is a port file. If the file is a port file and the "dontwait" <open option> is not specified, the program is suspended until buffers are available to perform the write. If the "dontwait" <open option> is specified, the program is not suspended when buffers are not available. Instead, control returns to the program without performing the write. The success or failure of the operation can be determined by examining the associated I/O result. If an exception occurs and the I/O result is not handled programmatically, the program is terminated. Refer to "I/O Exception Handling" for information concerning I/O results.

The "urgent" clause is meaningful only when the Transmission Control Protocol/Internet Protocol (TCP/IP) is being used. This clause sets the "urgent indication" with the data. For more information on TCP/IP, refer to the "BNA Host Services Operations Guide."

The use of <subfile variable>s, <station variable>s, and the "dontwait" and "urgent" <open option>s are Unisys extensions to ANSI Pascal.

See also

I/O Exception Handling. 253

PUT BYTES PROCEDURE

<put_bytes procedure>

```

-- put_bytes -- ( ----<file variable>----- , --<io length>----->
                |-----|
                |-<subfile variable>-|
                |-----|
                |-<station variable>-|
                |-----|
>----- ) -----|
|-----|
| - , --<put option>-|

```

<io length>

--<integer expression>--|

The <put_bytes procedure> allows a program to do a variable-length write from a file's <buffer variable>. The <put_bytes procedure> writes <io length> bytes of the file's buffer variable to the specified file. If <io length> exceeds either the length of the file's buffer variable or the MAXRECSIZE of the file, the smaller number of bytes is written.

NOTE

Actually, an I/O which is <io length> FRAMESIZE UNITS will be requested. The compiler will set FRAMESIZE to 8 for all files to which this procedure can be applied.

As with the <get_bytes procedure>, whether a variable-length I/O is actually performed depends upon the values of the BLOCKSTRUCTURE, SIZEVISIBLE, and KIND file attributes. For further information, see the description of the <get_bytes procedure> and the example programs below.

The <component type> of the file must be an <explicit array type> or <explicit record type>.

The <put option> is the same as the <put option> parameter defined for the <put procedure>. It is not allowed if the first parameter is a <station variable>.

Real Expression

The write operation treats variables of the <real type> and <fixed-point type> similarly. If the <frac digit> is provided, the number is written in fixed-point format; if it is not, the number is written in floating-point format. The default <field width> for a <real expression> is 15 characters.

In floating-point format, the number contains the following components:

1. A sign ("-") if the number is negative, blank if it is positive)
2. The first significant digit (or 0, if the number is 0)
3. A decimal point (.)
4. The fractional part (at least one digit)
5. The exponent symbol (E)
6. The sign of the exponent ("+" or "-")
7. Two digits of the exponent

If the <field width> specified is smaller than the minimum number of characters necessary to represent the number, the <field width> specification is ignored, and the number is written with one digit of fractional part. If the specified <field width> is larger, the number is expanded by adding trailing zeros to the fractional part.

In fixed-point format, the number will contain the following components:

1. A minus sign (-) if the number is negative
2. The integral part of the number (trunc(<real expression>))
3. A decimal point (.)
4. The <frac digit> of the fractional part of the number

If a specified <field width> is smaller than the minimum number of characters necessary to represent the number in fixed-point format, the <field width> specification is ignored and the entire number is written, including the <frac digit> of the fractional part. If the specified <field width> is larger, the number is written with leading blanks. If the number of significant digits requested is fewer than the number of significant digits in the system's representation of the number, the number will be rounded at the last digit written.

A space for a sign character is reserved for numbers in fixed-point format. If the number is negative, the minus sign is placed in that space. If the number is positive, the plus sign is not displayed; however, the space for the sign character is still reserved and a blank is displayed. Because of this characteristic, the extra space should be considered when output for fixed-point numbers is being formatted.

Examples

Statement -----	Result -----
write(f,1.2345:20)	" 1.23450000000000E+00"
write(f,-27.1828e-3:14)	"-2.7182800E-02"
write(f,0.31:3)	" 3.1E-01"
write(f,-96E12:7)	"-9.6E+13"
write(f,0.317269:3)	" 3.2E-01"
write(f,-965E12:7)	"-9.7E+14"
write(f,0.31726e7:7:3)	"3172600.000"
write(f,-965E12:1:7)	"-965000000000000.0000000"
write(f,0.31726e7:13:3)	" 3172600.000"
write(f,-965E-2:12:7)	" -9.6500000"
write(f,3.1776e-1:13:3)	" 0.318"
write(f,-962.5E-2:12:2)	" -9.63"

See also

Eof Function	274
Variables	407
Write Procedure	308

Predefined Procedures and Functions

TRUNC FUNCTION

<trunc function>

```
-- trunc -- ( ---<real expression>----- ) --|
                |                               |
                |-<fixed-pt expression>-|
```

The <trunc function> returns the integer value, computed by truncation, of the specified <real expression>. If the result is greater than "maxint" or less than "-maxint", an error occurs.

If the nearest integer to the <fixed-pt expression> is greater than "maxint" or less than "-maxint", a double-precision integer value is returned. It is an error if the absolute value of the integer nearest to the <fixed-pt expression> is greater than the double-precision integer value.

Example

Statement	Result
-----	-----
trunc(3.5)	3
trunc(-3.5)	-3

5.8 GENERAL PROCEDURES AND FUNCTIONS

<general procedure>

```

----<abort procedure>-----|
|<accept procedure>-----|
|<date procedure>-----|
|<display procedure>-----|
|<getattribute procedure>-|
|<move_bytes procedure>---|
|<programdump procedure>--|
|<setattribute procedure>-|
|<time procedure>-----|
|<wait procedure>-----|

```

<general function>

```

----<elapsedtime function>-----|
|<iotime function>-----|
|<linenumber function>----|
|<odd function>-----|
|<pred function>-----|
|<runtime function>-----|
|<succ function>-----|
|<versionoption function>-|

```

Many general procedures and functions are extensions to ANSI Pascal that allow the program access to system-specific features, such as file attributes, the program's accumulated run time, I/O time, and elapsed time, the interface to the Operator Display Terminal (ODT), and the system's time and date values. Other general procedures and functions listed here are part of ANSI Pascal and provide features not described elsewhere in this manual.

IOTIME FUNCTION

<iotime function>

```
-- iotime --|
```

The <iotime function> returns, as a real value in units of seconds, the total I/O time that has been charged to the program.

The <iotime function> is a Unisys extension to ANSI Pascal.

LINENUMBER FUNCTION

<linenumber function>

-- linenumber --|

The <linenumber function> returns, as an integer value, the sequence number of the source file record where the function is used. The integer value returned by the <linenumber function> is in the range 0 through 99999999. This value is extracted from columns 73 through 80 of the record.

Example

```
PROCEDURE Assert( value1, value2, line_nbr : INTEGER );
BEGIN
IF value1 <> value2 THEN
    write( tracefile,
          'Assertion failure at line number ',
          line_nbr, '.', ' Value 1 = ', value1,
          ', Value 2 = ', value2, '.'
        );
END; {Assert}
...
Assert( linklibrary( GeneralSupportLib ),
        libres( SuccessfulLink ),
        linenumber
      );
...
```

This program fragment checks to ensure that a link to the library "GeneralSupportLib" was successful. If not, the program writes out the message and sends the line number of the record where the link was unsuccessful.

Predefined Procedures and Functions

MOVE BYTES PROCEDURE

<move_bytes procedure>

```
-- move_bytes -- ( --<byte count>-- , --<source>-- , ----->
>-<source offset>-- , --<destination>-- , --<destination offset>---->
>- ) -----|
```

<byte count>

<source offset>

<destination offset>

```
--<integer expression>--|
```

<source>

```
----<explicit array variable>----|
|                               |
|-<explicit record variable>-|
```

<destination>

```
----<explicit array variable>----|
|                               |
|-<explicit record variable>-|
```

The <move-byte procedure> moves contiguous bytes from one place to another. It takes <byte count> contiguous bytes from <source>, starting at the <source offset> byte, and moves the contiguous bytes to <destination>, starting at the <destination offset> byte. Both <source offset> and <destination offset> begin at 1 (that is, they are 1 relative).

If the <move_bytes procedure> is used with an overlapping source and destination, the result is undefined. For example, if the first byte of the destination is the last byte of the source, the last byte moved may have the value of the source's first byte instead of its original value. If the source and destination must overlap, do the move in two steps.

The <move_bytes procedure> is a Unisys extension to ANSI Pascal.

Predefined Procedures and Functions

PROGRAMDUMP PROCEDURE

<programdump procedure>

```

-- programdump -----|
|                       |
|   |<----- , -----|   |
|   |                   |   |
| - ( ---<programdump option>--- ) -|

```

<programdump option>

```

----- ARRAY -----|
|                   |
| - ARRAYS -----|
|                   |
| - BASE -----|
|                   |
| - CODE -----|
|                   |
| - FILE -----|
|                   |
| - FILES -----|
|                   |
| - LIBRARIES -----|
|                   |
| - PRIVATELIBRARIES -|
|                   |
| - ALL -----|

```

The <programdump procedure> calls a procedure in the Master Control Program (MCP) that writes the contents of the program stack to the taskfile of the program. The <programdump option> specifies the items of the stack that are to be analyzed and written. The information produced by the PROGRAMDUMP call is written to the file specified by the TASKFILE task attribute of the program. Although task attribute access is not yet available in Pascal, the TASKFILE file attributes can be file equated at run time.

If no <programdump option> is specified, the stack is analyzed and printed according to the specifications in the OPTION task attribute of the program. If the contents of the program arrays are to be printed, the option ARRAY or ARRAYS must be specified.

The bottom (or "base") of the program stack is printed if the BASE option is specified. The MCP uses a portion of each stack to contain various words needed to control, identify, and log the program.

The segment dictionary of the program is printed if the CODE option is specified. The actual code is printed only for segments that are referenced by the program at the time of the PROGRAMDUMP call. Value arrays in the segment dictionary are printed when both the CODE option and either the ARRAY or ARRAYS option are specified.

If the program files are to be printed and analyzed, the FILE or FILES option must be specified. As each file is encountered, each word of the File Information Block (FIB) is separately named and, in some cases, analyzed.

The LIBRARIES option causes the stacks of all libraries being used by the program to be printed. The PRIVATELIBRARIES option causes the stacks of all private libraries being used by the program to be printed.

Specifying the ALL option is equivalent to specifying all the other options.

Example

```
ProgramDump;
```

```
ProgramDump( array, files );
```

```
IF IdNote.NextLink <> NIL THEN  
  BEGIN  
    display('*** Program xyz taking program dump ***');  
    ProgramDump( all );  
  END;
```

The first use of the <programdump procedure> analyzes and writes the program stack information according to whatever specifications were in the OPTION task attribute of the program.

The second use of the <programdump procedure> analyzes the arrays and files associated with the program.

The third use of the <programdump procedure> analyzes all information about the program and writes the information to the task file.

Predefined Procedures and Functions

VERSIONOPTION FUNCTION

<versionoption function>

-- versionoptionfunction --|

The <versionoption function> returns to a <versionoption type> record the version, cycle, and patch numbers defined by the most recent VERSION compiler control option. The integer values returned are <integer> type and are in the following ranges:

<u>Parameter</u>	<u>Range</u>
<release>	0..99
<cycle>	0..999
<patch>	0..9999

The <versionoption type> is a predefined identifier.

Example

```

$VERSION 36.186.1986

PROGRAM version_example( output );
...
VAR
  version_nbr      : versionoptiontype;
  year             : 0..9999;
  month           : 1..12;
  day             : 1..31;
  hour            : 0..23;
  minute          : 0..59;
  second          : 0..59;
BEGIN
  ...
  version_nbr :=versionoption;
  date( year, month, day );
  time( hour, minute, second );
  ...
  writeln( output, 'Compiled with sample test version',
           version_nbr.release : 1, '.',
           version_nbr.cycle : 1, '.',
           version_nbr.patch : 1,
           { the ' : 1' writes the numbers in the smallest number of
             { spaces possible }
           ' on ', month : 1, '/', day : 1, '/', year : 1,
           ' at ', hour : 1, ':', minute : 1, '.'
         );
  { Using the "with" statement...}
  with version_nbr do
    writeln( output, 'Compiled with sample test version',
            release : 1, '.', cycle : 1, '.', patch : 1, '.'):
  ...
END.

```

This program fragment obtains the release number, the cycle number, and the patch number assigned at the beginning of the program and writes those numbers to the output file. In this example, the release number is 36, the cycle number is 186, and the patch number is 1986.

SUBFILE VARIABLE

<subfile variable>

--<file variable>-- (-- subfile --<subfile index>--) --|

<subfile index>

--<integer expression>--|

A port file (a file having PORT as its KIND attribute) can have one or more associated "subfiles," each of which can be connected to a different complementary subfile. A <subfile variable> can be used instead of a <file variable> in certain predefined procedures and functions to refer to a particular subfile. In all cases, the type of the <subfile variable> is the same as the type of the <file variable> that it qualifies. Note also that all subfiles share the buffer variable of their associated <file variable>.

For more information on the use of subfiles in Pascal, refer to "Open Procedure," "Close Procedure," "Get Procedure," and "Put Procedure" under "File Handling Procedures and Functions." Also refer to "Wait Procedure," "Getattribute Procedure" and "Setattribute Procedure" under "General Procedures and Functions."

The <subfile variable> is a Unisys extension to ANSI Pascal.

See also

File Handling Procedures and Functions.	230
General Procedures and Functions.	386

TEMPLATE VARIABLE

A <template variable> is a <variable> declared as a <template type>.

TEXTFILE VARIABLE

A <textfile variable> is an <entire variable> declared as a <textfile type>.

VERSIONOPTION VARIABLE

A <versionoption variable> is a <variable> declared as a <versionoption type>.

VLSTRING VARIABLE

A <vlstring variable> is a <variable> declared as a <vlstring type>.

X-BOOL VARIABLE

An <x-bool variable> is a <variable> declared as an <x-bool type>.

X-INTEGER VARIABLE

An <x-integer variable> is a <variable> declared as an <integer type>.

X-NUM VARIABLE

An <x-num variable> is a <variable> declared as an <x-num type>.

X-REAL VARIABLE

An <x-real variable> is a <variable> declared as an <x-real type>.

X-WORD VARIABLE

An <x-word variable> is a <variable> declared as an <x-num type>.

Interpretation of Program Text

Reserved Word

<reserved word>

AND	DIV	FOR	INTERFACE	NOT	PROGRAM	TYPE
ARRAY	DO	FROM	LABEL	OF	RECORD	UNTIL
BEGIN	DOWNTO	FUNCTION	LIBRARY	OR	REPEAT	USAGE
CAND	ELSE	GOTO	MOD	OTHERWISE	SET	VAR
CASE	END	IF	MODULE	PACKED	THEN	WHILE
CONST	EXPORT	IMPORT	NIL	PROCEDURE	TO	WITH
COR	FILE	IN				

The <reserved word>s are language keywords that cannot be redefined by the programmer. In general, these are words the compiler uses to recognize declarations, statements, and operators.

Predefined Identifier

<predefined identifier>

abort	dispose	log	scan_neq
abs	elapsedtime	mark	seek
accept	eof	max	setattribute
addstation	eoln	maxint	sfixed
adjust_template	erf	min	sin
arccos	exp	move_bytes	sinh
arcsin	false	new	skiptochannel
arctan	filevalue	odd	sqr
arctanh	fillchar	open	sqrt
attach_template	fixed	ord	string
Boolean	fixeddiv	output	succ
Boolean1	freeze	pack	tan
Boolean4	gamma	page	tanh
cancel	get	pos	template
char	get_bytes	pred	text
chr	getattribute	programdump	time
close	hex	put	true
concat	input	put_bytes	trunc
cos	insert	random	u_display
cosh	integer	read	unpack
cotan	integer48	readln	update
date	integer96	real	versionoption
delete	iores	real48	versionoptiontype
deletestation	iotime	release	wait
delinklibrary	length	reset	word48
dfhvalue	libraryvalue	rewrite	word96
digits	libres	round	write
digits_s	linenumber	runtime	writeln
display	linklibrary	s_digits	z_display
display_s	ln	s_display	
display_z	lngamma	scan_eq1	

<Predefined identifier>s are <identifier>s that have a predefined meaning in Pascal; as with user-defined <identifier>s, <predefined identifier>s can be redefined, but the former definition becomes unavailable within the scope of the redefinition.

MERGE OPTION

<merge option>

```

-- MERGE -----|
      |           |
      |- " --<file title>-- " -|

```

(Type: Boolean, Default: FALSE)

The MERGE option, when enabled, merges source language records from the primary input file (CARD) with source language records from the secondary input file (SOURCE). The <file title>, if present, causes the specified file to be read as the SOURCE file (the <file title> is used to assign the TITLE attribute of the SOURCE file). If no <file title> is specified, SOURCE is assumed. The single quotation mark (') can be substituted for the double quotation mark (") when delimiting the <file title>, but it must be used in pairs.

This option remains enabled throughout a compilation and cannot be disabled. Subsequent attempts to SET or RESET this option are treated as errors and are ignored.

This option can appear only on a compiler control record that is placed before any Pascal source record in the CARD file.

NEW OPTION

<new option>

```

-- NEW -----|
      |           |
      |- " --<file title>-- " -|

```

(Type: Boolean, Default: FALSE)

The NEW option, when enabled, creates a new source language symbolic (NEWSOURCE) of all source language records accepted for compilation. This new symbolic includes any input records that were omitted by the OMIT option and any permanent compiler control records, but it does not include any records that were discarded by enabling the DELETE option or the VOID option.

The <file title>, if present, causes the specified file to be written as the NEWSOURCE file (the <file title> is used to assign the TITLE attribute of the NEWSOURCE file). If no <file title> is specified, NEWSOURCE is assumed. The single quotation mark (') can be substituted for the double quotation mark (") when delimiting the <file title>, but it must be used in pairs.

This option remains enabled throughout a compilation and cannot be disabled. Subsequent attempts to SET or RESET this option are treated as errors and are ignored.

See also

DELETE Option	446
OMIT Option	457
VOID Option	469

NOBOUNDS OPTION

<nobounds option>

```
-- NOBOUNDS --|
```

(Type: Boolean, Default: FALSE)

The NOBOUNDS option, when enabled, eliminates all range-checking code that the compiler would normally generate to verify the assignment compatibility of set variables, subrange variables, and vlstring variables. Range checking performed by the system, such as the range check on an array index when the array is an <entire variable> (not part of a record), cannot be disabled.

See also

Variables	407
---------------------	-----

repetitive statement

A structured statement that specifies that its subcomponent statements are to be repeated.

reserved word

A word that has special meaning within a programming language and that generally cannot be redefined or redeclared by the programmer.

run time

The time during which an object code file is executed. Run time is also referred to as "execution time."

same type

A term applied to two types when they have been defined as equivalent to each other or both equivalent to a third type.

schema

An outline or plan.

In Pascal, an array declaration that includes one or more dynamic discriminants. In other words, a schema is a type of incomplete array declaration. Using an array schema as a formal parameter makes it possible to pass arrays with different bounds and different numbers of elements to the same formal parameter.

schemata

The plural of schema.

sequential statement

A structured statement in which the subcomponent statements are executed in the order in which they appear, without conditions or repetitions.

simple statement

A statement that performs a single action and contains no other statements.

source program

A program coded in a language one or more steps removed from machine language. It must be translated into machine language before use.

standard character set

The character set used by the compiler to represent the "char" type and string types. The standard character set is EBCDIC unless changed to ASCII using the STRINGS compiler control option.

structured statement

A statement that can have statements as subcomponents.

structured type

A data type that defines variables that can have multiple components.

substring

A portion of a string or vlstring variable.

successor

In an ordinal type, the value having the next highest ordinal number.

TCP/IP

See "Transmission Control Protocol/Internet Protocol"

Transmission Control Protocol/Internet Protocol (TCP/IP)

A host-to-host data communications protocol. In conjunction with BNA, Version 2, the Unisys TCP/IP software product provides a link to incompatible systems across a TCP/IP network, using the familiar BNA interface.

BIBLIOGRAPHY

- Advanced Data Dictionary System (ADDS) User's Guide (form 1180551). Unisys Corporation.
- IEEE Standard Pascal, American National Standard. Computer Programming Language ANSI/IEEE, 770X3.97-1983.
- Binder Programming Reference Manual (form 5014582). Unisys Corporation. Formerly Binder Reference Manual.
- BNA Architectural Description Reference Manual, Volume 1 (form 1132172). Unisys Corporation.
- BNA Host Services Operations Guide (form 1170339). Unisys Corporation.
- BNA Version 1 Program Agent Programming Guide (form 1169653). Unisys Corporation.
- CANDE Operations Reference Manual (form 1169869). Unisys Corporation.
- Editor User's Guide (form 1169976). Unisys Corporation.
- I/O Subsystem Programming Reference Manual (form 1169984). Unisys Corporation.
- Message Translation Utility User's Guide (form 1169554). Unisys Corporation.
- Operator Display Terminal (ODT) System Commands Operations Reference Manual (form 1169612). Unisys Corporation.
- Pascal Programming Reference Manual, Volume 2: Product Interfaces (form 1170107). Unisys Corporation.
- Pascal User Manual and Report. K. Jensen and N. Wirth. New York: Springer-Verlag, 1978.

PASCAL PROGRAMMING REFERENCE MANUAL

System Software Utilities Operations Reference Manual (form 1170024).
Unisys Corporation.

Work Flow Language (WFL) Programming Reference Manual (form 1169802).
Unisys Corporation.

Data representation (cont.)

- sets, 494
- simple types, 483
- structured types, 489
- subranges, 488
- template types, 115
- textfiles, 494
- undefined operands, 496
- versionoption types, 119
- vlstrings, 495
- x-bool types, 120
- x-integer types, 134
- x-num types, 122
- x-real types, 131
- x-word types, 132

Data types, 483

<date procedure>, 389

<day>, 389

<declaration part>, 39

Declarations, 39

- label, 46
- library, 142
- procedure, 145
- variable, 139

<declared function>, 186

<declared procedure>, 174

Definitions, 135

- array schema, 135
- constant, 47
- type and schema, 68

Definitions and declarations, 39

<delete index>, 358

<delete length>, 358

<delete option>, 446

<delete procedure>, 358

<deletestation procedure>, 272

<delinklibrary procedure>, 348

<destination>, 396A

<destination offset>, 396A

Device-dependent textfile attributes, 248

<dfhvalue function>, 273

<digit>, 423

Digits

- x-num type, 123

Digits_s

- x-num type, 125

<directive>, 145

<discriminant identifier>, 136

<discriminant identifier list>, 135

<discriminant specification>, 135

<discriminant value>, 85

<discriminated array schema>, 85

DISK attribute, 234

- <disk file header attribute>, 427
- <display procedure>, 391
- Display_s
 - x-num type, 130
- Display_z
 - x-num type, 128
- <dispose procedure>, 341
- DIV, 206
 - integer expression, 206
 - real expression, 210
- Division operators, 185
- DO, 166
- <domain type>, 102
- Dontcare, library usage, 18
- Dontwait option
 - used in <close procedure>, 270
 - used in <open procedure>, 285
- DOWNTO, 166
- <duration option>, 18
- <dynamic allocation procedure>, 337
- <dynamic variable>, 411
- Dynamic variables, 340

- EBCDIC and ASCII character sets, 497
- <elapsedtime function>, 392
- <element type>, 86
- ELSE, 172
- Empty statement, 160
- END, 159
- End-of-line markers, 243
- <ending index>, 222
- <ending seq number>, 449
- <entire variable>, 408
- Entire variables, 408
- Entry points, referencing, 21
- <enumerated constant>, 93
- <enumerated constant definition>, 48
- <enumerated constant identifier>, 49
- <enumerated expressions>, 197
- <enumerated type>, 93
- <enumerated type identifier>, 70
- <enumerated variable>, 414
- <eof function>, 274
- <eoln function>, 275
- <erf function>, 374
- Error, detection and reporting, 517
- <errorlimit option>, 447
- <errorlist option>, 447
- <event>, 281
- <event-valued file attribute>, 427
- <exp function>, 374
- <explicit array type>, 74

Index

<explicit array variable>, 414
 <explicit record type>, 74
 <explicit record variable>, 415
 <explicit type>, 72, 74
 <exponent part>, 425
 Exponentiation operators, 185
 Exponentiation semantics, table of, 212
 Export declaration, 41
 Exported functions and procedures, 20
 <expression>, 181
 Expression concepts, 183
 Expressions, 181

- arithmetic, 183
- bit pattern, 189
- Boolean, 192
- char, 195
- enumerated, 197
- integer, 206
- operands, 184
- ordinal, 183
- pointer, 208
- real, 210
- relational, 213
- set, 219
- string, 221
- vlstring, 222
- x-bool, 224
- x-num, 225
- x-real, 226
- x-word, 227

 Expressions by type, 189
 Extensions to ANSI Pascal, 515
 <external directive>, 145
 EXTERNAL directive, 34
 <external file identifier>, 4
 <external file specification>, 4
 EXTERNAL module

- binding, 34

Factorial, 3
 Features, implementation-defined, 510
 Features, implementation-dependent, 513
 <field designator>, 410
 <field identification>, 104
 <field identifier>, 56, 104
 Field identifiers, redefining in embedded records, 8
 <field list>, 104
 <field list value>, 56
 <field location>, 105
 <field type>, 105
 <field value>, 56
 <field width>, 309

- <field-designated constant>, 62, 64
- Fields, 106
- <file attribute assignment>, 400
- <file attribute request>, 393
- File attributes, 234
 - and mnemonic values, 427
- <file handling function>, 230
- <file handling procedure>, 230
- <file title>, 449
- <file type>, 95
- <file type identifier>, 70
- <file variable>, 415
- File-structuring attributes, 242
- Files
 - compiler, 473
 - logical, 234
 - permanent, 235
 - physical, 234
 - port, 248
 - representation of, 491
 - standard and textfiles, 233
 - temporary, 235
- FILESTATE attribute, 285
- FILEUSE, 250
- <filevalue function>, 276
- <fillchar procedure>, 359
- <final value>, 166
- <fixed part>, 104
- <fixed-part value>, 56
- <fixed-point constant definition>, 48
- <fixed-point constant identifier>, 49
- Fixed-point expressions, 199
- Fixed-point format, 313
- <fixed-point type>, 97
 - fixed, 97
 - sfixed, 98
- <fixed-point type identifier>, 70
- <fixed-point type transfer function>, 321
- <fixed-point variable>, 415
 - option in a <read textfile procedure>, 296
- <fixed-pt expression>
 - write textfile procedure usage, 312
- <fixeddiv function>, 374
- Floating-point format, 313
- FOR, 166
- <for statement>, 166
- <formal discriminant part>, 135
- <formal parameter list>, 153
- Forms field, 464
- Forms mode, 464
- Forward, 145
- Forward declaration, function, 150
- Forward declaration, procedure, 146

Index

FORWARD directive, 34
 <frac digit>, 309
 FRAMESIZE attribute, 242
 <freeze procedure>, 349
 <function declaration>, 149
 Function descriptions, 268
 <function designator>, 186
 <function identifier>, 149
 Function, declared forward, 150
 <functional parameter>, 154
 Functionname, 142
 FUNCTIONNAME attribute, 143

<gamma function>, 375
 <general function>, 386
 <general procedure>, 386
 Generation and inspection mode, 233
 Generation mode operations
 on standard files, 239
 on textfiles, 246
 Generic parameter types, 23
 <get option>, 277
 <get procedure>, 277
 <get_bytes procedure>, 279
 <getattribute procedure>, 393
 Global activation record, 16
 <goto statement>, 169

<happened function>, 281
 Heap, 18
 <heapsize option>, 448
 <hex constant>, 190
 <hex digit>, 190
 <hex type>, 100
 <hex type identifier>, 70
 Host type, 113
 <hours>, 403

I/O exception handling, 253
 I/O results, 253
 close, 257
 get, put, read, readln, write, writeln, 256
 open, 253
 open on port files, 255
 <identifier>, 423
 Identifiers, 7
 COMS, 434
 context-sensitive, 433
 miscellaneous, 434
 predefined, 432

Identifiers (cont.)
 rules for defining, 9
 valid and invalid, 424
<if statement>, 172
<immediate option>, 440
IMPLEMENTATION directive, 34
Implementation-defined features, 510
Implementation-dependent features, 513
Import Declaration, 44
Importing entry points, 39
<inclnew option>, 448
<include option>, 449
<index constant>, 62, 63
<index expression>, 409
<index type>, 85
<indexed array variable>, 409
<indexed variable>, 409
<indexed vlstring variable>, 409
<initial value>, 166
<insert index>, 360
<insert procedure>, 360
Inspection and generation mode, 233
Inspection mode operations
 on standard files, 237
 on textfiles, 243
<integer constant>, 206
<integer constant definition>, 48
<integer constant identifier>, 49
Integer division, 185
<integer expression>, 206
<integer operator>, 206
<integer primary>, 206
Integer remainder division, 185
<integer type>, 101
<integer type identifier>, 70
<integer type transfer function>, 323
<integer variable>, 297
 compare format for <real variable>, 415
<integer-valued file attribute>, 427
INTERFACE directive, 34
<interface function declaration>, 20
<interface part>, 20
<interface procedure and function declarations>, 20
<interface procedure declaration>, 20
Interpretation of program text, 429
INTMODE attribute, 242
<intname>, 449
Intname, 142
<io length>, 290
<iores function>, 282
<ioresult mnemonic>, 282
<iotime function>, 395
<ipccapable option>, 451

Index

KIND attribute, 234

<label>, 46
 <label declarations>, 46
 Labels, 7
 rules for defining, 9
 Lazy I/O, 246
 <length function>, 361
 <letter>, 423
 <lib mnemonic>, 142
 Libaccess, 142
 LIBACCESS attribute, 143
 Libparameter, 142
 Libraries
 entry points of, 13
 handling procedures and functions of, 345
 initiating, 13
 matching option, 19
 multiple users of, 18
 parameter matching of, 21
 referencing entry points, 21
 restrictions of, 14
 sharing option, 16
 structure of, 1
 <library>, 13
 <library attribute assignment>, 400
 <library attribute phrase>, 142
 <library attribute request>, 393
 <library declarations>, 142
 <library handling function>, 345
 <library handling procedure>, 345
 <library heading>, 13
 <library identifier>, 13
 <library result mnemonic>, 354
 <libraryvalue function>, 350
 <libres function>, 354
 <lineinfo option>, 451
 <linenumber function>, 396
 Lines, 243
 <linkage option>, 452
 <linklibrary procedure>, 351
 <list option>, 452
 <listdollar option>, 453
 <listimport option>, 453
 <listincl option>, 453
 <listomitted option>, 454
 <ln function>, 376
 <lngamma function>, 376
 <local directive>, 145
 Local variables, 11
 <log function>, 376
 Logical files, 234

Mantissa, 487
<map option>, 454
<mark procedure>, 342
Mark value, 339
Marked variables, 342
<matching option>, 19
<max function>, 377
<maximum length>, 118
MAXINT, 521
MAXRECSIZE attribute, 242
<member designator>, 219
<merge option>, 455
<min function>, 379
<minutes>, 403
<miscellaneous identifier>, 434
<mnemonic value>, 427
<mnemonic-valued file attribute>, 427
<mnemonic-valued library attribute>, 142
MOD, 206
 integer expression, 206
 real expression, 210
<module>, 33
<module body>, 34
<module directive>, 433
<module heading>, 34
<module identifier>, 34
<month>, 389
<move_bytes procedure>, 396A
Multiplication operators, 185
MYUSE attribute, 250

<new array type>, 85
<new bits type>, 90
<new enumerated type>, 93
<new file type>, 95
<new hex type>, 100
<new option>, 455
New option, <open procedure> usage, 284
<new pointer type>, 102
<new procedure>, 343
<new record type>, 104
<new set type>, 111
<new subrange type>, 113
<new vlstring type>, 118
NEWFILE attribute, 250
Nil, 102
<nobounds option>, 456
<nonapostrophe character>, 422
Norewind option, 270
NOT, 192
<noxreflist option>, 457
NUL character, 388

Index

Null reference, 208
 <number>, 425
 <number of bits>, 90
 <number of hex digits>, 100

<odd function>, 397
 Offer option, <open procedure> usage, 285
 <offset change>, 367
 <omit option>, 457
 On-top-of variables, 338
 OPEN file attributes, 401
 <open option>, 283
 <open procedure>, 283
 Operands, undefined, 496
 Operators
 addition, 185
 binary, 184
 Boolean NOT, 185
 DIV, 185, See also DIV
 division, 185
 exponentiation, 185
 MOD, 185, See also MOD
 multiplication, 185
 relational, 185
 subtraction, 185
 unary, 184
 Operators, precedence of, 184
 <option expression>, 442
 <option phrase>, 441
 <option primary>, 442
 Options, 443
 <ord function>, 324
 <ordinal expression>, 183
 <ordinal relation>, 215
 <ordinal type>, 73
 <ordinal type identifier>, 105
 <array schema definition>, 136
 declared as predefined type, 325
 <ordinal type transfer function>, 325

<pack procedure>, 326
 PACKED, 489
 <packed array variable>, 326
 Packed arrays, 489
 <page option>, 458
 <page procedure>, 287
 Parameter list congruity, 158
 Parameter matching, 21
 Parameter type matching, 23
 Parameter types, Pascal and generic, 23
 <parameterized bit pattern type>, 72

Partial word, 481
Pascal, 7
Pascal parameter types, 23
Pascal, compared with ANSI Pascal, 509
<patch number>, 467
<pattern>, 362
Permanent files, 235
Physical files, 234
<pointer expression>, 208
<pointer relation>, 216
<pointer type>, 72, 102
<pointer type identifier>, 70
<pointer variable>, 415
Pointer-valued attributes, 251
Pointers, 496
POP, 441
Port files, 248
 I/O results, 255
 open options, 285
 subfiles of, 417
<pos function>, 362
<pos source>, 362
Precedence of operators, 184
Precedence, order of, 442
<pred function>, 398
<predefined function>, 229
<predefined identifier>, 432
<predefined procedure>, 229
Predefined procedures and functions, 229
Presence bit, 460
Presence-bit interrupts, 460
PRESENT attribute, 250
Private, 17
<procedural parameter>, 154
<procedure and function declarations>, 144
Procedure and function descriptions, 268
<procedure declaration>, 145
Procedure descriptions, 268
<procedure identifier>, 145
<procedure invocation statement>, 174
Procedure, declared forward, 146
<program>, 2
Program, 1
 block, 7
 scope of, 7
 structure of, 1
<program heading>, 2
<program identifier>, 2
<program parameters>, 4
<program text>, 430
Program text, interpretation of, 429
<program unit>, 1
PROGRAMDUMP PROCEDURE, 398A

PROTECTION file attribute, 5
 Purge option, 270
 <put option>, 288
 <put procedure>, 288
 <put_bytes procedure>, 290

Railroad diagrams, explanation of, 525
 <random function>, 381
 <read parameter>, 295
 <read procedure>, 294
 <read textfile procedure>, 295
 <read-only parameters>, 154
 <readln procedure>, 302
 <real constant>, 210
 <real constant definition>, 48
 <real constant identifier>, 49
 <real expression>, 210
 format of, 313
 <real primary>, 210
 <real type>, 103
 <real type identifier>, 70
 <real type transfer function>, 328
 Real values, range of, 521
 <real variable>, 415
 read textfile procedure usage, 298
 <real-valued file attribute>, 427
 Real48, 131
 <record boundary>, 438
 <record constant>, 64
 <record constant identifier>, 52
 <record type>, 104
 <record type identifier>, 70
 <record value>, 56
 <record variable>, 415
 Record variables, 10
 Records, 491
 Referencing library entry points, 21
 <rel op>, 213
 <relational expression>, 213
 Relational expressions, symbol chart of, 213
 Relational operators, 185
 Relations
 arithmetic, 214
 bit pattern, 214
 ordinal, 215
 pointer, 216
 set, 217
 string, 218
 <release number>, 466
 <release procedure>, 344
 Remote files, 416
 <repeat statement>, 176

Representation of standard files, 242
Reserve option, 270
<reserved word>, 431
RESET, 441
<reset procedure>, 303
<result type>, 149
<result type ID>, 374
<returned length>, 279
<rewrite procedure>, 304
<round function>, 382
<runtime function>, 399

S_digits

x-num type, 124

S_display

x-num type, 129

Same types, 78

Save option, 270

<scale factor>, 97

<scan function>, 363

<scan set>, 363

<scan target>, 363

Scan_eq1, 363

Scan_neg, 363

Schema and type concepts, 71

Schema definitions, 68

<schema discriminant>, 188

Scope, 7

<seconds>, 403

<seek procedure>, 305

Semantics, exponentiation, 212

<sequence base option>, 459

<sequence increment option>, 460

<sequence number>, 469

<sequence option>, 458

SET, 441

<set constant expression>, 59

<set constant identifier>, 52

<set constant primary>, 59

<set constructor>, 219

<set expression>, 219

<set operator>, 219

<set primary>, 219

<set relation>, 217

<set type>, 111

<set type identifier>, 70

<set variable>, 416

<setattribute procedure>, 400

Sets, 494

Sharedbyall, 18

Sharedbyrununit, 16

<sharing option>, 16

Index

- <simple constant definition>, 48
- <simple constant identifier>, 48
- Simple statements, 159
- <simple type>, 71
- <sin function>, 383
- <sinh function>, 383
- SIZEVISIBLE attribute, 290
- <skiptochannel procedure>, 306
- <source>, 396A
- <source offset>, 396A
- <special token>, 436
- <sqr function>, 383
- <sqrt function>, 384
- Stack operations, 11
- Standard files, 237
- Standard files and textfiles, 233
- <starting index>, 222
- <starting seq number>, 449
- <statement>, 159
- <statement list>, 159
- <statement part>, 159
- <station index>, 416
- <station variable>, 416
- <statistics option>, 460
- <string constant>, 221
- <string constant definition>, 48
- <string constant identifier>, 49
- <string expression>, 221
- <string function>, 365
- <string handling function>, 356
- <string handling procedure>, 356
- <string relation>, 218
- <string type>, 89
- <string type identifier>, 89
- <string variable>, 416
 - textfile procedure usage, 299
- <string-valued file attribute>, 427
- <string-valued library attribute>, 142
- <strings option>, 461
- <structured constant>, 52
- Structured constant
 - array value, 54
 - record value, 56
- <structured constant definition>, 52
- <structured constant identifier>, 52
- Structured statements, 159
- <structured type>, 72
- Structured types, 489
 - arrays, 489
 - files, 491
 - packed arrays, 490
 - packed sets, 494
 - pointers, 496

Structured types (cont.)

- records, 491
- sets, 494
- textfiles, 494
- undefined operands, 496
- vlstrings, 495
- <subfile index>, 417
- <subfile variable>, 417
- SUBFILEERROR attribute, 285
- <subrange type>, 113
- <subrange type identifier>, 70
- Subranges of data types, 488
- <substring designator>, 222
- <substring expression>, 222
- <substring length>, 222
- <succ function>, 402
- Syntax diagrams, explanation of, 525
- System-specific information, 521

Tag field, 106

- <tag field identifier>, 56
- <tag value>, 56
- <tan function>, 384
- <tanh function>, 384

Tape file options

- norewind, 270
- purge option, 270
- reserve, 270

- <target option>, 462

TCP/IP, See Transmission Control Protocol/Internet Protocol

- <template domain type>, 115
- <template handling procedure>, 366
- <template type>, 73
- type and schema definitions, 115

- <template type identifier>, 70

- <template variable>, 417

Temporary files, 235

Temporary, default value, 18

Terminology, 233

- <textfile type>, 117

- <textfile type identifier>, 70

- <textfile variable>, 417

Textfiles, 243

- input and output, 249
- representation of, 247

THAW command, 14

- duration option usage, 18

Threatening statement, 167

- <time procedure>, 403

Title, 142

TITLE attribute, 143

TITLE attribute, length of, 521

Index

- TO, 166
- <token>, 430
- <token separator>, 437
- Transfer functions
 - bit pattern type, 318
 - Boolean type, 319
 - fixed-point, 321
 - integer type, 323
 - real type, 328
 - x-bool type, 331
 - x-num type, 332
 - x-real type, 334
 - x-word type, 335
- Transmission Control Protocol/Internet Protocol, 289
- <trunc function>, 385
- <ttywrite option>, 463
- <type>, 71
- Type and schema concepts, 71
- <type and schema definitions>, 68
- Type compatibility, 78
- <type definition>, 68
- Type descriptions, 85
 - array, 85
 - bits, 90
 - Boolean, 91
 - char, 92
 - enumerated, 93
 - file, 95
 - hex, 100
 - integer, 101
 - pointer, 102
 - real, 103
 - record, 104
 - set, 111
 - string, 89
 - subrange, 113
 - template, 115
 - textfiles, 117
 - variable-length string, 118
 - versionoption, 119
 - x-bool, 120
 - x-integer, 134
 - x-num, 122
 - x-real, 131
 - x-word, 132
- <type identifier>, 76
- Type transfer facilities, 317
- Type transfer procedures and functions, 316
- Types, explicit, 74
- Types, ordinal, 73

- U_display
 - x-num type, 126
- Unary operator, 184
- Undefined operands, 496
- Undefined variables, 419
- <underscore>, 423
- <unpack procedure>, 329
- <unpacked array variable>, 326
- Unpacked arrays, 489
- <unsigned integer>, 425
- <unsigned number>, 425
- <unsigned real>, 425
- Update option, <open procedure> usage, 285
- <update procedure>, 307
- Urgent clause, 289
- <usage clause>, 16
- <user option>, 465
- Userdebug option, 465

- <value option>, 440
- <value parameter>, 153
- <value parameter type>, 153
- VAR, 139
- <variable declarations>, 139
- <variable identifier>, 139
- <variable identifier list>, 139
- <variable parameter>, 153
- <variable parameter type>, 153
- Variable-length (vlstring) string types, 118
- Variables, 407
 - by access, 407
 - by types, 414
 - undefined, 419
- <variant list element>, 106
- <variant part>, 105
- <variant selector>, 105
- <variant specifier>, 343
- <variant-part value>, 56
- VERSION attributes, 254
- <version option>, 466
- Version option, 466
- <versionoption function>, 404A
- <versionoption type>, 68, 119
- <versionoption variable>, 418
- <vlstring constant expression>, 61
- <vlstring constant identifier>, 52
- <vlstring expression>, 222
 - write textfile procedure usage, 312
- <vlstring type>, 118
- <vlstring type identifier>, 70
- <vlstring variable>, 418
 - read textfile procedure usage, 300

Index

Vlstrings, structured cypes of, 495
 <void option>, 469

Wait option, open procedure usage, 284
 <wait procedure>, 405
 <warnsupr option>, 470
 <while statement>, 177
 <with statement>, 178
 WITH statement, scope of field identifiers, 10
 <write parameter>, 309
 <write procedure>, 308
 <write textfile procedure>, 309
 <writeln procedure>, 315

<x-bool expression>, 224
 <x-bool type>
 Boolean1, 120
 Boolean4, 120
 type and schema concepts, 120
 <x-bool type identifier>, 70
 <x-bool type transfer function>, 331
 <x-bool variable>, 418
 <x-integer type>
 integer48, 134
 integer96, 134
 type and schema concepts, 134
 <x-integer variable>, 418
 <x-num expression>, 225
 <x-num type>
 type and schema concepts, 122
 X-num type
 binary(n), 122
 binary(n,s), 123
 digit(n), 123
 digit(n,s), 123
 digits_s(n), 125
 digits_s(n,s), 125
 display_s(n), 130
 display_s(n,s), 130
 display_z(n), 128
 display_z(n,s), 128
 s_digits(n), 124
 s_digits(n,s), 124
 s_display(n), 129
 s_display(n,s), 129
 u_display(n), 126
 u_display(n,s), 126
 z_display(n), 127
 z_display(n,s), 127
 <x-num type identifier>, 70
 <x-num type transfer function>, 332

- <x-num variable>, 418
- <x-real expression>, 226
- <x-real type>
 - real48, 131
 - type and schema concepts, 131
- <x-real type identifier>, 70
- <x-real type transfer function>, 334
- <x-real variable>, 418
- <x-word expression>, 227
- <x-word type>
 - type and schema concepts, 132
 - word48, 132
 - word96, 132
- <x-word type identifier>, 70
- <x-word type transfer function>, 335
- <x-word variable>, 418
- <xref option>, 470
- <xreffiles option>, 471
- <xreflater option>, 472

<year>, 389

Z_display

- x-num type, 127

NOTES

NOTES

NOTES

