American National Standard

Adopted for Use by the Federal Government

FIPS PUB 68-2

See Notice on Inside Front Cover

for information systems –

programming languages – full BASIC

ANSI X3.113-1987

american national standards institute, inc.
1430 broadway, new york, new york 10018

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to reaffirm, revise, or withdraw this standard no later than five years from the date of approval. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

This standard has been adopted for Federal Government use.

Details concerning its use within the Federal Government are contained in Federal Information Processing Standards Publication 68-2, BASIC. For a complete list of the publications available in the Federal Information Processing Standards Series, write to the Standards Processing Coordinator (ADP), Institute for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD 20899.

Published by

American National Standards Institute
1430 Broadway, New York, New York 10018

Printed in the United States of America

PC1½M1287/36

ANSI®
X3.113-1987

American National Standard for Information Systems –

Programming Languages – Full BASIC

Secretariat

Computer and Business Equipment Manufacturers Association

Approved January 28, 1987

American National Standards Institute, Inc

Abstract

This standard presents the form for and the interpretation of programs written in the BASIC programming language for use on computers and information processing systems.

Foreword

(This Foreword is not part of American National Standard X3.113-1987.)

This American National Standard specifies the form for and the interpretation of programs written in the BASIC programming languages for use on computers and information processing systems. Its purpose is to promote portability of BASIC programs for use on a variety of machines.

This standard was approved as an American National Standard by the American National Standards Institute on January 28, 1987.

Suggestions for improvement of this standard will be welcome. They should be sent to the Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Processing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

Edward Lohse, Chair
Richard Gibson, Vice-Chair
Catherine A. Kachurik, Administrative Secretary

Organization Represented	*Name of Representative*
American Express	D. L. Seigal
	Lucille Durfee (Alt)
American Library Association	Paul Peters
American Nuclear Society	Geraldine C. Main
	D. R. Vondy (Alt)
AMP Incorporated	Patrick E. Lannan
	Edward Kelly (Alt)
Association for Computing Machinery	Kenneth Magel
	Jon A. Meads (Alt)
Association of the Institute for Certification of Computer Professionals	Thomas M. Kurihara
AT&T Communications	Henry L. Marchese
	Richard Gibson (Alt)
AT&T Technologies	Herbert V. Bertine
	Paul D. Bartoli (Alt)
Burroughs Corporation	Ira R. Purchis
	Stanley Fenner (Alt)
Control Data Corporation	Charles E. Cooper
	Keith Lucke (Alt)
Cooperating Users of Burroughs Equipment	Thomas Easterday
	Donald Miller (Alt)
Data General Corporation	John Pilat
	Lyman Chapin (Alt)
Data Processing Management Association	Christian G. Meyer
	Ward Arrington (Alt)
	Terrance H. Felker (Alt)
Digital Equipment Computer Users Society	William Hancock
	Dennis Perry (Alt)
Digital Equipment Corporation	Gary S. Robinson
	Delbert L. Shoemaker (Alt)
Eastman Kodak	Gary Haines
	Charleton C. Bard (Alt)
General Electric Company	William R. Kruesi
General Services Administration	William C. Rinehuls
	Larry L. Jackson (Alt)
GUIDE International	Frank Kirshenbaum
	Sandra Swartz Abraham (Alt)
Harris Corporation	Walter G. Fredrickson
	Rajiv Sinha (Alt)
Hewlett-Packard	Donald C. Loughry
Honeywell Information Systems	Thomas J. McNamara
	David M. Taylor (Alt)

Organization Represented	*Name of Representative*
IBM Corporation	Mary Anne Gray
	Robert H. Follett (Alt)
IEEE Computer Society	Sava I. Sherr
	Thomas M. Kurihara (Alt)
	Thomas A. Varetoni (Alt)
Lawrence Berkeley Laboratory	David F. Stevens
	Robert L. Fink (Alt)
Moore Business Forms	Delmer H. Oddy
National Bureau of Standards	Robert E. Rountree
	James H. Burrows (Alt)
National Communications System	George W. White
NCR Corporation	Thomas W. Kern
	A. Raymond Daniels (Alt)
Perkin-Elmer Corporation	Christopher Beling
	Russ Lombardo (Alt)
Prime Computer, Inc	Joseph Schmidt
	John McHugh (Alt)
Railinc Corporation	R. A. Petrash
Recognition Technology Users Association	Herbert F. Schantz
	G. W. Wetzel (Alt)
SHARE, Inc	Thomas B. Steel
	Robert A. Rannie (Alt)
Sperry Corporation	Marvin W. Bass
	Jean G. Smith (Alt)
Texas Instruments, Inc	Presley Smith
	Richard F. Trow, Jr (Alt)
3M Company	J. Wade Van Valkenburg
Travelers Insurance Companies, Inc.	Joseph T. Brophy
U.S. Department of Defense	Fred Virtue
	Belkis Leong-Hong (Alt)
VIM	Chris Tanner
	Madeleine Sparks (Alt)
VISA U.S.A.	Jean T. McKenna
	Susan Crawford (Alt)
Wang Laboratories, Inc.	Marsha Hayek
	Joseph St. Amand (Alt)
Xerox Corporation	John L. Wheeler
	Roy Pierce (Alt)

Technical Committee X3J2 on BASIC developed this standard. The following X3J2 members attended at least three committee meetings during the development phase of the standard.

J. M. Totton, Chair
T. E. Kurtz, Past Chair
J. Cugini, Past Vice Chair
S. Garland, Past Vice Chair
J. A. N. Lee
Past Vice Chair
B. Zino, Secretary
G. J. Hornik
Past Secretary

N. Abel
S. Auditore
T. Bair
D. B. Bearisto
R. E. Bruce
G. M. Bull
M. D. Busch
G. A. Cook
D. Denman
F. Dollak
D. Dudley
M. O. Duke
D. E. Elliott
J. K. Elliott
D. M. Esbensen
J. L. Evans
C. French
D. E. Gilsinn
G. K. Haas
H. H. Hall
I. T. Hardy
J. A. Harle
D. Hedges

R. E. Holzman
M. Hui
G. L. Isaacs
C. G. Jeans
H. Kaikow
N. Kareemi
T. D. Keerl
B. Kenner
A. Klossner
M. Koo
M. Kovalick
J. B. Lane
H. D. Leeds
J. J. Leicht
D. Levine
D. A. Lillie
A. Luehrmann
D. S. Martin
S. Miller
R. A. Miner
L. D. Montgomery
R. Nelson
D. A. Nobles

B. G. Noparstak
R. Onasch
D. Petersen
C. Phillips
L. Piazza
D. Pitsch
T. Powderly
J. Raskin
A. Riccomi
W. Roberts
R. D. Ross
R. D. Shurtleff, Jr
G. H. Smith
L. Stabile
D. H. Su
D. E. Taylor
S. Taylor
J. E. Tuecke
K. Warner
W. Watson
W. M. Wedel

Contents

SECTION PAGE

Tables

SECTION PAGE

American National Standard for Information Systems –

Programming Languages – Full BASIC

1. Scope, Purpose, and Referenced and Related Standards

1.1 Scope. This standard establishes:

(1) The syntax of programs written in BASIC, including "core" BASIC and various extensions thereto

(2) The formats of data and the minimum precision and range of numeric representations and the minimum length and set of characters in strings that are acceptable as input to an automatic data processing system being controlled by a program written in BASIC

(3) The formats of data and the minimum precision and range of numeric representations and the minimum length and set of characters in strings that can be generated as output by an automatic data processing system being controlled by a program written in BASIC

(4) The semantic rules for interpreting the meaning of a program written in BASIC

(5) The errors and exceptional circumstances that shall be detected and also the manner in which such errors and exceptional circumstances shall be handled

In addition, this standard contains an optional module containing editing facilities for BASIC programs.

Although the BASIC language was originally designed primarily for interactive use, this standard describes a language that is not so restricted. This standard is not meant to

preclude the use of any particular implementation technique; for example, interpreters or incremental or one-pass compilers.

1.2 Purpose. This standard is designed to promote the interchangeability of BASIC programs among a variety of automatic data processing systems. Programs conforming to this standard will be said to be written in American National Standard (ANS) BASIC.

1.3 Referenced and Related Standards

1.3.1 Referenced American National Standard. This standard is to be used in conjunction with the American National Standard for Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986. When this standard is superseded by a revision approved by the American National Standards Institute, Inc., the revision shall apply.

1.3.2 Related Standards. The standards listed here are for information only and are not essential for the completion of the requirements of this standard:

ANSI X3.30-1985, Information Systems - Representation for Calendar Date and Ordinal Date for Information Interchange

ANSI X3.42-1975, The Representation of Numerical Values in Character Strings for Information Interchange

ANSI X3.43-1986, Information Systems - Representation of Local Time of Day for Information Interchange

ANSI X3.60-1978, Minimal BASIC

ANSI X3.124-1985, Information Systems - Computer Grapshics - Graphical Kernel System (GKS), Functional Description

ANSI/IEEE 726-1982, Real-time BASIC for CAMAC

ISO 7942-1985, Information Processing - Computer Graphics - Graphical Kernel System (GKS), Functional Description

2. Conformance

This standard is organized in a modular fashion; conformance to it is defined with respect to particular sets of the following seven modules:

(1) A core module, which encompasses all programs whose syntax conforms to Sections 4 through 12, excluding those portions of Section 11 that describe enhanced files.

(2) Two enhanced files modules, one for internal-format records and one for native-format records, each of which encompasses all programs whose syntax conforms to the productions so indicated in Section 11, together with the core.

(3) A graphics module, which encompasses all programs whose syntax conforms to Section 13 together with the core.

(4) A real-time module, which encompasses all programs whose syntax conforms to Section 14 together with the core.

(5) A fixed decimal module, which encompasses all programs whose syntax conforms to Section 15 together with the core.

(6) An editing module, which encompasses all unsorted programs and editing commands whose syntax conforms to Section 16.

There are two aspects of conformance to a set of modules in this standard: conformance by a program written in the BASIC language, and conformance by an implementation which processes such programs. Broadly speaking, the conformance requirements are structured so that any program conforming to a set of modules will produce the same results when executed by any implementation conforming to the same or an encompassing set of modules (though certain implementation-dependent features are noted in Appendix C).

2.1 Program Conformance.

A program conforms to a set of modules in this standard only when:

(1) The program and each statement or other syntactic element contained therein is syntactically valid according to the syntactic rules specified by this standard as belonging to that set

(2) The program as a whole violates none of the global constraints imposed by this standard on the application of the syntactic rules

2.2 Implementation Conformance. An implementation conforms to a set of modules in this standard only when:

(1) It accepts and processes all programs conforming to that set of modules in this standard

(2) It reports reasons for rejecting any program which does not conform to that set of modules in this standard

(3) It interprets errors and exceptional circumstances according to the specifications of this standard

(4) It interprets the semantics of each statement of a conforming program according to the specifications in this standard

(5) It interprets the semantics of a conforming program as a whole according to the specifications in this standard

(6) It accepts as input, manipulates, and can generate as output numbers of at least the precision and range specified in this standard

(7) It accepts as input, manipulates, and can generate as output strings of at least the length and composed of at least those characters specified in this standard

(8) It is accompanied by documentation available to the user that describes the actions taken in regard to features referred to as "undefined" or "implementation-defined" in this standard

(9) It is accompanied by documentation available to the user that describes and identifies all enhancements to the language defined in this standard

This standard makes no requirement concerning the interpretation of the semantics of any statement or program as a whole that does not conform to this standard.

In addition, an implementation conforms to the editing requirements of this standard if it accepts and processes

unsorted programs and editing commands according to the specifications in Section 16.

2.3 Errors. This standard does not include specific requirements for reporting syntactic errors in the text of a program. Implementations conforming to a set of modules in this standard may accept programs written in an enhanced language without having to report all constructs not conforming to that set of modules.

Whenever a statement, or other program element, does not conform to the syntactic rules given herein, and that statement, or program element, does not have a clear, well-documented implementation-defined meaning, an error shall be reported. Errors shall be reported in a clear and well-documented way, and whenever feasible the implementation should indicate the erroneous statement and the position of the error within the statement.

2.4 Exceptions. An exception is a circumstance arising in the course of execution of a program when an implementation recognizes that the semantic rules of this standard cannot be followed or that some resource constraint is about to be exceeded. All exceptions described in this standard shall be detected, reported, and processed when they occur, unless some mechanism provided in 12.1 or in an enhancement to this standard has been invoked by the user to handle exceptions.

In the absence of programmer-specified recovery procedures, exceptions shall be handled by the recovery procedures specified in this standard. If no recovery procedure is specified in this standard, or if restrictions imposed by the hardware or the operating environment make it impossible to follow the procedure specified in this standard, then the way in which the exception is handled depends on the context. If the exception occurred in an invocation of a function, picture, or subprogram, then the exception is "propagated back" to the invoking statement in the invoking program unit (see 12.1). If this propagation procedure reaches the main-program or a parallel-section, or if the exception occurred in the main-program or a parallel-section, then the exception shall be handled by terminating the program or, in the case of real-time-programs, the parallel-section, generating the exception.

The way in which the default exception handling mechanism reports an exception is implementation-defined, except that the

contents of the report shall identify at least the original exception code and the line number of the line in which the original exception occurred.

Except in the case of files, when several exceptions are caused by the execution of a single statement of a program this standard does not specify an order in which these exceptions shall be detected, reported, or processed.

If an implementation determines that a particular statement in a conforming program will always cause an exception when executed, the implementation may issue a warning to the user. Nonetheless, the implementation shall accept and execute the program, according to the normal semantic rules specified herein.

3. Syntax Specification and Definitions

3.1 Method of Syntax Specification. The syntax, through a series of rewriting rules known as "productions," defines syntactic objects of various types, such as program or expression, and describes which strings of symbols are objects of these types.

In the syntax, upper-case-letters, digits, and (possibly hyphenated) lowercase words are used as "metanames," i.e., as names of syntactic objects. Most of these metanames are defined by productions in terms of other metanames. In order that this process terminate, certain metanames are designated as "terminal" metanames, and productions for them are not included in the syntax. With the exception of the construct "[implementation-defined]," all terminal metanames occur for the first time and are defined in 4.1. It should be noted in particular that all upper-case-letters are terminal metanames that generally denote both themselves and their lowercase equivalents (except in the productions defining upper-case- and lower-case-letters, in which the letters denote only themselves). The digits are terminal metanames that denote themselves. In addition, the construct "[implementation-defined]" is not a unique syntactic object, but each occurrence of it is defined by each implementation in an appropriate fashion for the object in question. In some cases a recommendation as to the representation of the object is given in the corresponding remarks section.

We illustrate further details of the syntax by considering some examples from 5.1. The production

```
fraction = period integer
```

indicates that a fraction is a period followed by an integer. Since "period" is a terminal metaname (i.e., it does not occur on the left-hand side of any production), the semantics in 4.1 identify the particular character denoted by a period.

What is an integer? The production

```
integer = digit digit*
```

indicates that an integer is a digit followed by an arbitrary number of other digits. An asterisk is a syntactic operator indicating that the object it follows may be repeated any number of times, including zero times.

What is a digit? In 4.1 the production

```
digit  = 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9
```

indicates that a digit is either a 0, a 1, ..., or a 9. The slant is a syntactic operator meaning "or" and is used to indicate that a metaname can be rewritten in one of several ways. Since the digits are terminal metanames, our decipherment of the syntax for a fraction comes to an end. The semantics in 4.1 identify the digits in terms of the characters they represent.

A question-mark is a syntactic operator like the asterisk, indicating that the object it follows may be omitted. For example, the production

```
exrad = E sign? integer
```

indicates that an exrad consists of the letter E or e followed by an optional sign followed by an integer.

Parentheses may be used to group sequences of metanames together. For example,

```
variable-list  = variable (comma variable)*
```

defines a variable-list to consist of a variable followed by an arbitrary number of other variables separated by commas. If we want parentheses actually to appear in syntactic objects, rather than just want to use them to describe syntactic objects, then we indicate their presence by the metanames "left-parenthesis" and "right-parenthesis."

When several syntactic operators occur in the same production, the following order of precedence is employed. The operators "?" and "*" apply only to the word or parenthesized expression they immediately follow. The operator "/" applies to the sequence of words and expressions, separated by spaces, which occur since the beginning of the entire expression, the last "/", or the last unmatched left parenthesis. Thus, for example,

```
significand  =  integer period? / integer? fraction
```

is equivalent to

```
significand  =  (integer (period)?) /
                ((integer)? fraction)
```

Spaces in the syntax are used to separate terms in a production from each other. Special conventions are observed regarding spaces in BASIC programs (see 4.1).

Some syntactic objects are defined by more than one production. For example, in 5.2 we find

simple-variable > simple-numeric-variable

and in 6.2 we find

simple-variable > simple-string-variable.

Those two productions are equivalent to the single production below (provided no other definition of simple-variable exists)

```
simple-variable = simple-numeric-variable /
                  simple-string-variable
```

In all cases, a greater-than-sign is used in place of an equals-sign to indicate a multiple definition; such definitions are equivalent to a single definition containing the various right-hand sides separated by slants.

As an illustration of the method of syntax specification, following is a description of the syntax of this method. The terminal metanames occurring below are defined in 4.1.

```
 1. production         = metaname spaces
                         (equals-sign / greater-than-sign)
                         spaces syntax-expression
 2. metaname           = lower-case-letter metacharacter*
 3. metacharacter      = lower-case-letter / hyphen
 4. spaces             = space* end-of-line? space* space
 5. syntax-expression  = syntax-term
                         (spaces? slant spaces? syntax-term)*
 6. syntax-term        = syntax-factor (spaces syntax-factor)*
 7. syntax-factor      = syntax-primary repetition?
 8. syntax-primary     = metaname / digit digit* /
                         upper-case-letter upper-case-letter* /
                         left-parenthesis space*
                         syntax-expression space*
                         right-parenthesis
 9. repetition         = asterisk / question-mark
```

3.2 Definitions. For the purpose of this standard, the following terms have the meanings indicated.

BASIC, A term applied as a name to members of a special class of languages that possess similar syntaxes and semantic

meanings; acronym for Beginner's All-purpose Symbolic Instruction Code.

batch-mode, The processing of programs in an environment where no provision is made for user interaction.

can, The term used in a descriptive sense to indicate that standard-conforming programs are allowed to contain certain constructions and that standard-conforming implementations are required to process such programs correctly.

end-of-line, The characters or indicators that identify the termination of a line. Lines of three kinds may be identified in BASIC: program lines, print lines, and input-reply lines. End-of-lines may vary between the three cases and may also vary depending upon context. Thus, for example, the end-of-line in an input-reply may vary on a given system depending on the source for input being used in interactive or batch mode.

Typical examples of end-of-line are carriage-return, carriage-return line-feed, and end-of-record (such as end-of-card).

error, A flaw in the syntax of a program that causes the program to be incorrect.

exception, A circumstance arising in the course of executing a program when an implementation recognizes that the semantic rules of this standard cannot be followed or that some resource constraint is about to be exceeded. Certain exceptions (nonfatal exceptions) may be handled by automatic recovery procedures specified in this standard. These and other exceptions may also be handled by recovery procedures specified in the program (cf. 12.1). If no recovery procedure is given in this standard (fatal exceptions) or if restrictions imposed by the hardware or operating environment make it impossible to follow the given procedure, and if no recovery procedure is specified in the program, then the way in which the exception is handled depends on the context. If the exception occurred in an invocation of a function, picture, or subprogram, then the exception is "propagated back" to the invoking statement of the invoking program unit (see 12.1). If this propagation procedure reaches the main-program or a parallel-section, or if the exception occurred in the main-program or a parallel-section, then the exception shall be handled by terminating the program or, in the case of real-time-programs, the parallel-section, generating the exception.

external, With respect to procedures, this term refers to a procedure lexically not contained within a larger program-unit.

identifier, A character string used to name a variable, an array, an array-value, an exception-handler, a function, picture, subprogram, or a program. In a real-time-program, identifiers are also used to name parallel-sections, events, structures, process-ports, data-ports, and message-ports.

interactive mode, The processing of programs in an environment that permits the user to respond directly to the actions of individual programs and to control the initiation and termination of these programs.

internal, With respect to record-type, this term refers to data representations such that both the type and exact value of the written data are preserved and retrievable by subsequent read operations. With respect to procedures, this term refers to a procedure lexically contained within a larger program-unit and sharing data with that unit.

keyword, A character string, usually with the spelling of a commonly used or mnemonic word, that provides a distinctive identification of a statement or a component of a statement of a programming language.

Table 1 lists all the keywords defined in this standard. Because of the modular nature of the standard, some keywords may be required in certain modules, but not in others. Following each keyword is a code indicating which modules contain that keyword.

Table 1. BASIC Keywords

c = core r = real-time
f = enhanced files d = fixed decimal
g = graphics e = editing

ACCESS	c	AND	c	ANGLE	cg
AREA	g	ARITHMETIC	cd	ARRAY	g
ASK	cg	AT	ge	BASE	c
BEGIN	c	BREAK	c	CALL	c
CASE	cr	CAUSE	c	CELLS	g
CHAIN	c	CHOICE	g	CLEAR	g
CLIP	g	CLOSE	c	COLLATE	cf
COLOR	g	CONNECT	r	CONTINUE	c
DATA	c	DATUM	c	DEBUG	c

DECIMAL	c	DECLARE	cr	DEF	cd
DEGREES	c	DELAY	r	DELETE	fe
DEVICE	g	DIM	c	DISCONNECT	r
DISPLAY	c	DO	c	DRAW	g
ELAPSED	c	ELSE	c	ELSEIF	c
END	cgr	ERASABLE	c	ERASE	c
EVENT	r	EXCEPTION	c	EXIT	cgr
EXTERNAL	cgd	EXTRACT	e	FILETYPE	c
FIRST	e	FIXED	d	FOR	c
FROM	r	FUNCTION	cd	GET	gr
GO	c	GOSUB	c	GOTO	c
GRAPH	g	HANDLER	c	HEIGHT	g
IF	c	IMAGE	c	IN	cgr
INPUT	cr	INTERNAL	c	IS	c
JUSTIFY	g	KEY	f	KEYED	f
LAST	e	LENGTH	c	LET	c
LIMIT	g	LINE	cg	LINES	g
LIST	e	LOCATE	g	LOOP	c
MARGIN	c	MAT	cfg	MESSAGE	r
MISSING	c	MIX	g	MULTIPOINT	g
NAME	c	NATIVE	cf	NEXT	c
NOT	c	NUMERIC	cfrd	OF	fr
OFF	c	ON	cr	OPEN	c
OPTION	c	OR	c	ORGANIZATION	c
OUT	r	OUTIN	cr	OUTPUT	cr
PARACT	r	PARSTOP	r	PICTURE	g
PIXEL	g	PLOT	g	POINT	g
POINTER	c	POINTS	g	PORT	r
PRINT	c	PROCESS	r	PROGRAM	c
PROMPT	c	PUT	r	RADIANS	c
RANDOMIZE	c	RANGE	g	READ	c
RECEIVE	r	RECORD	f	RECSIZE	c
RECTYPE	c	RELATIVE	f	REM	c
RENUMBER	e	REST	c	RESTORE	c
RETRY	c	RETURN	c	REWRITE	f
SAME	c	SEIZE	r	SELECT	cr
SEND	r	SEQUENTIAL	c	SET	c
SETTER	c	SHARED	r	SIGNAL	r
SIZE	cg	SKIP	cf	standard	cf
START	r	STATUS	g	STEP	ce
STOP	c	STREAM	c	STRING	cfr
STRUCTURE	r	STYLE	g	SUB	c
TAB	c	TEMPLATE	f	TEXT	g
THEN	c	THERE	c	TIME	r
TIMEOUT	cr	TO	cgre	TRACE	c
UNTIL	c	URGENCY	r	USE	c
USING	cg	VALUE	g	VARIABLE	c
VIEWPORT	g	WAIT	r	WHEN	c

WHILE	c	WINDOW	g	WITH	cfg
WRITE	c	ZONEWIDTH	c		

Keywords may also be spelled using lowercase letters or mixed uppercase and lowercase letters.

line, Two types of lines are described in the standard, a physical line and a logical line. A physical line is an ordered sequence of characters that terminates with an end-of-line. A physical line starts with a line-number or with an ampersand. A logical line consists of a line-number followed by an ordered sequence of text in which each line-continuation has logically been replaced by a space.

machine infinitesimal, The smallest positive value (other than zero) that can be represented and manipulated by a BASIC implementation.

may, The term used in a permissive sense to indicate that a standard-conforming implementation may or may not provide a particular feature.

native, With respect to record-type, this term refers to a record with a specified structure for the fields within the record, so as to be compatible with records generated by other languages on the same system. With respect to (numeric or string) data, this term refers to data for which certain semantic rules are left implementation-defined (e.g., collating sequence, precision) so as to be directly implementable on the host hardware.

overflow, With respect to numeric operations, the term that is applied to the condition that exists when a prior operation has attempted to generate a result whose magnitude exceeds MAXNUM (cf. 5.4.4), or that exceeds the maximum value that can be represented by the declared format of a fixed point variable or array.

With respect to string operations, the term that is applied to the condition that exists when a prior operation has attempted to generate a result that has more characters than can be contained in a string of maximal length, as determined by the language processor.

With respect to string assignment, the term that is applied to the condition that exists when a prior operation has attempted

to assign a value that is longer than the declared or default maximum of a string-variable or string-defined-function.

print zone, A contiguous set of character positions in a printed output line which may contain an evaluated print-statement element.

program unit, A self-contained part of a BASIC program consisting either of the main-program, which is the sequence of lines up to and including the line containing an END statement, or of an external-sub-def, external-function-def, external-picture-def, or parallel-section.

reserved word, A character string whose usage as a routine-, string-, or numeric-identifier is forbidden in a BASIC program. These words are:

(1) The no-argument supplied function names: DATE, EXLINE, EXTYPE, MAXNUM, PI, RND, TIME, TRANSFORM, DATE$, and TIME$

(2) The identifiers used in array-values: CON, IDN, ZER, and NUL$

(3) The keywords: NOT, ELSE, PRINT, and REM

rounding, The process by which a representation of a value with lower precision is generated from a representation of higher precision taking into account the value of that portion of the original number that is to be omitted. For example, rounding X to the nearest integer may be accomplished by INT(X+0.5) (cf. 5.4).

shall, The term that is used in an imperative sense to indicate that a program is required to be constructed, or that an implementation is required to act, as specified in order to meet the constraints of standard conformance.

significant digits, The contiguous sequence of digits between the high-order nonzero digit and the low-order digit, without regard for the location of the radix point. Commonly, in a normalized floating point internal representation, only the significant digits of a representation are maintained in the significand. In fixed-point representation, the low order digit is the rightmost one explicitly specified, and nonsignificant high order digits may be maintained.

truncation, The process by which a representation of a value with lower precision is generated from a representation of higher

precision by merely deleting the unwanted low-order digits of the original representation.

underflow, With respect to numeric operations, the term applied to the condition that exists when a prior operation has attempted to generate a result, other than zero, that is less in magnitude than machine infinitesimal.

4. Program Elements

A BASIC program is a sequence of lines containing statements. Each line is itself a sequence of characters.

4.1 Characters

4.1.1 General Description

The character set for BASIC shall be the character set as described in ANSI X3.4-1986 (ASCII).

4.1.2 Syntax

1.	character	= quotation-mark / non-quote-character
2.	quoted-string-character	= double-quote / non-quote-character
3.	non-quote-character	= ampersand / apostrophe / asterisk / circumflex-accent / colon / comma / dollar-sign / equals-sign / exclamation-point / greater-than-sign / left-parenthesis / less-than-sign / number-sign / percent-sign / question-mark / right-parenthesis / semicolon / slant / underline / unquoted-string-character
4.	double-quote	= quotation-mark quotation-mark
5.	unquoted-string-character	= space / plain-string-character
6.	plain-string-character	= digit / letter / period / plus-sign / minus-sign
7.	digit	= 0/1/2/3/4/5/6/7/8/9
8.	letter	= upper-case-letter / lower-case-letter
9.	upper-case-letter	= A/B/C/D/E/F/G/H/I/J/K/L/M/ N/O/P/Q/R/S/T/U/V/W/X/Y/Z
10.	lower-case-letter	= a/b/c/d/e/f/g/h/i/j/k/l/m/ n/o/p/q/r/s/t/u/v/w/x/y/z
11.	other-character	= [implementation-defined]

The syntax as described generates programs that contain no spaces other than those occurring in remark-strings, in certain quoted-strings, unquoted-strings, and literal-strings, or where the presence of a space is explicitly indicated by the metaname space.

Special conventions shall be observed regarding spaces. With the following exceptions, spaces may occur anywhere in a BASIC program without affecting the execution of that program and may be used to improve the appearance and readability of the program. Spaces shall not appear:

(1) Immediately preceding the line-number of a line
(2) Within line-numbers
(3) Within keywords
(4) Within identifiers
(5) Within numeric-constants
(6) Within multicharacter relation symbols

In addition, spaces that appear in quoted-strings, unquoted-strings, and format-strings shall be significant (though spaces that precede or follow an unquoted-string are not part of that string).

All keywords in a program, when used as such, shall be preceded and followed by some character other than a letter, digit, underline, or dollar-sign. A keyword may also be followed by an end-of-line.

4.1.3 Examples

None.

4.1.4 Semantics

The letters shall be the set of upper-case and lower-case Roman letters contained in the ASCII character set in positions 4/1 through 5/10 and 6/1 through 7/10, respectively.

The digits shall be the set of Arabic digits contained in the ASCII character set in positions 3/0 through 3/9.

The remaining characters shall correspond to the remaining graphic characters in positions 2/0 through 2/15, 3/10 through 3/15, 5/14, and 5/15 of the ASCII character set.

The names of the characters are specified in Table 1. Table 1 shall apply when the standard collating sequence is in effect, either by default or by explicit use of a COLLATE option (cf. 6.4, 6.6, and 8.1). The coding for the native collating sequence shall be implementation-defined.

All characters other than letters denote themselves. Letters denote themselves within quoted-strings, unquoted-

strings, and line-input-replies. Corresponding upper-case- and lower-case-letters shall be equivalent when used in identifiers and keywords. Quoted-string-characters also denote themselves, except for the double-quote, which denotes one occurrence of the quotation-mark in the value of the string.

4.1.5 Exceptions

None.

4.1.6 Remarks

Other-characters may be defined by an implementation to be part of the character set for BASIC. These characters may be used in strings and may be accepted as characters in data supplied in response to a request for input or generated as the value of the CHR$ function (cf. 6.4). The effects of these other-characters are implementation-defined.

Programs written using other-characters (except for end-of-line characters) do not conform to this standard.

4.2 Programs, Lines, and Blocks

4.2.1 General Description

A BASIC program is a sequence of lines. Each line contains a unique line-number that facilitates program editing and serves as a label for the statement contained in that line.

A BASIC program is divided logically into a number of program-units. The first of these is the main-program, which is terminated by an end-line. Following the main-program may be zero or more external-sub-defs, external-function-defs, or external-picture-defs. A BASIC program may contain a series of parallel-sections, each of which is a separate program-unit.

Certain logical groupings of lines within a BASIC program are called blocks.

4.2.2 Syntax

```
 1. program                > program-name-line?  main-program
                             procedure-part*
 2. program-name-line      = line-number PROGRAM program-name
                             function-parm-list?  tail
 3. program-name           = routine-identifier
 4. main-program           = unit-block* end-line
 5. unit-block             = internal-proc-def / block
 6. internal-proc-def      > internal-function-def /
                             internal-sub-def /
                             detached-handler
 7. block                  > statement-line / loop /
                             if-block / select-block /
                             image-line / protection-block
 8. statement-line         = line-number statement tail
 9. line-number            = digit digit*
10. statement              > declarative-statement /
                             imperative-statement /
                             conditional-statement
11. declarative-statement  > data-statement /
                             declare-statement /
                             dimension-statement /
                             null-statement /
                             option-statement / remark-statement
12. imperative-statement   > array-assignment /
                             array-input-statement /
                             array-line-input-statement /
                             array-print-statement /
                             array-read-statement /
```

```
                                array-write-statement /
                                ask-statement /
                                break-statement / call-statement /
                                cause-statement / chain-statement /
                                close-statement / debug-statement /
                                erase-statement /
                                exit-do-statement /
                                exit-for-statement /
                                exit-function-statement /
                                exit-handler-statement /
                                exit-sub-statement /
                                gosub-statement / goto-statement /
                                handler-return-statement /
                                input-statement / let-statement /
                                line-input-statement /
                                numeric-function-let-statement /
                                open-statement / print-statement /
                                randomize-statement /
                                read-statement / restore-statement /
                                return-statement / set-statement /
                                stop-statement /
                                string-function-let-statement /
                                trace-statement / write-statement
13. stop-statement            = STOP
14. conditional-statement     = if-statement /
                                on-gosub-statement /
                                on-goto-statement
15. tail                      = tail-comment? end-of-line
16. end-of-line               = [implementation-defined]
17. end-line                  = line-number end-statement tail
18. end-statement             = END
19. procedure-part            = remark-line* procedure
20. procedure                 > external-function-def /
                                external-sub-def
21. remark-line               = line-number
                                (null-statement / remark-statement)
                                end-of-line
22. line                      > case-line / case-else-line /
                                do-line / else-line /
                                elseif-then-line /
                                end-function-line /
                                end-handler-line /
                                end-if-line / end-line /
                                end-select-line / end-sub-line /
                                end-when-line /
                                external-function-line /
                                external-sub-line /
                                for-line / handler-line /
```

```
                                internal-def-line /
                                internal-function-line /
                                internal-sub-line /
                                if-then-line / image-line /
                                loop-line / next-line /
                                program-name-line / remark-line /
                                select-line / statement-line /
                                use-line / when-line /
                                when-use-name-line
23. program-unit              > main-program / procedure
24. line-continuation         = ampersand space* tail ampersand
```

A program shall be composed of a sequence of lines. In the case of a non real-time-program, exactly one of these lines shall be an end-line; the lines up to and including this end-line constitute the main-program.

Line-number zero is not allowed; leading zeroes shall have no effect. Lines shall occur in ascending line-number order (cf. Section 16). All references to line-numbers within a program-unit shall be to line-numbers of lines within that program-unit. The number of digits in a line-number shall not exceed 5. The value of a line-number shall not exceed 50000.

The manner in which the end of a line is detected is determined by the implementation; e.g., the end-of-line may be a carriage-return character, a carriage-return character followed by a line-feed character, or the end of a physical record.

A physical line in a program shall contain at most 132 characters before each end-of-line indicator.

At any place in which a space may be used, except in quoted-strings, unquoted-strings, literal-strings, and remark-strings (cf. 4.1 and 4.3), a line-continuation may be substituted for a space with no effect other than that of the space it replaces.

Parameters in the program-name-line shall not be explicitly dimensioned or declared in the main-program, or first parallel-section of a real-time-program (see 14.1).

4.2.3 Examples

```
 2. 100 PROGRAM  Graphit       & ! This program draws a graph
    &            (x,           & ! x is x-coordinate
    &               y)           ! y is y-coordinate
15. 999 END
```

4.2.4 Semantics

The program-name-line is the operand of the chain-statement (cf. 9.3). The relationship between the program-name and the program-designator in a program executing a chain-statement is implementation-defined. Parameters in the program-name-line are evaluated as described in 9.1. Their scope is the main-program or the lexically first parallel-section (see 14.1). For a program executed in isolation, the program-name has no effect. The effect of a parameter-list in a program-name-line for a program executed in isolation is implementation-defined.

Lines in a program shall be executed in sequential order, starting with the first line, until

(1) Some other action is dictated by execution of a line
(2) An exception occurs (unless it is a nonfatal exception that is not handled by a user defined exception-handler)
(3) A chain-statement is executed
(4) A stop-statement or end-statement is executed

The end-statement shall serve both to mark the physical end of the main-program and to terminate execution of the program when encountered.

Execution of a stop-statement shall also cause termination of execution of the program.

4.2.5 Exceptions

None.

4.2.6 Remarks

References to nonexistent line-numbers in a program-unit are syntax errors. Implementations may therefore treat them as exceptions, if they are documented as such.

4.3 Program Annotation

4.3.1 General Description

BASIC programs may be annotated by comments at the end of program lines or by separate remark-statements.

4.3.2 Syntax

1. remark-statement = REM remark-string
2. remark-string = character*
3. null-statement = tail-comment
4. tail-comment = exclamation-point remark-string

Line-continuations shall not occur in remark-strings.

4.3.3 Examples

```
1. REM  FINAL CHECK
4. ! COMPUTE AVERAGE
```

4.3.4 Semantics

If the execution of a program reaches a line containing a remark-statement or null-statement, then it shall proceed to the next line with no other effect.

A tail-comment has no effect upon the execution of the line in which it occurs. The remark-string in the tail-comment serves solely as a comment about the line.

4.3.5 Exceptions

None.

4.3.6 Remarks

None.

4.4 Identifiers

4.4.1 General Description

Identifiers are used to name variables, arrays, array-values, functions, programs, subprograms, exception-handlers, and pictures. In a real-time-program, they are also used to name parallel-sections, events, structures, and ports (cf. Section 14).

4.4.2 Syntax

```
1.  identifier              > numeric-identifier /
                              string-identifier /
                              routine-identifier
2.  numeric-identifier      = letter identifier-character*
3.  identifier-character    = letter / digit / underline
4.  string-identifier       = letter identifier-character*
                              dollar-sign
5.  routine-identifier      = letter identifier-character*
```

An identifier shall contain at most 31 characters, including the dollar-sign in the case of a string-identifier.

A given numeric-identifier may name a simple-numeric-variable, a one-, two-, or three-dimensional numeric-array, a numeric-function, or a numeric-array-value, but not more than one of these in a program-unit. Likewise, a given string-identifier may name a simple-string-variable, a one-, two-, or three-dimensional string-array, a string-function, or a string-array-value, but not more than one of these in a program-unit.

A given identifier may name an internal-sub-def, an internal-function-def, a detached-handler, or an internal-picture-def, but not more than one of these in a program-unit.

A given routine-identifier shall not name more than one of an external-function-def, an external-sub-def, an external-picture-def, a main-program, or a parallel-section in a program.

A numeric-identifier that names an external-function-def may not be used as a routine-identifier.

The names of the no-argument supplied functions or array-values CON, DATE, EXLINE, EXTYPE, IDN, MAXNUM, PI, RND, TIME, TRANSFORM, and ZER shall not be used as numeric-identifiers to name any other entity. The names of the no-argument supplied functions or array-values DATE$, NUL$, and TIME$ shall not be

used as string-identifiers to name any other entity. The keywords NOT, ELSE, PRINT, and REM shall not be used as identifiers.

4.4.3 Examples

2. X
   sum
4. A$
   last_name$
5. INVERT

4.4.4 Semantics

Each program-unit is a distinct entity in that identifiers used to name variables, arrays, detached-handlers, internal-function-defs, internal-sub-defs, or internal-picture-defs (see 13.5) defined within program-units shall be local to each invocation of the program-unit in which they occur (i.e., they shall name different objects in different program-units and in different invocations of the same program-unit). Identifiers used to name supplied-functions or program-units, however, shall be global to the entire program (i.e., they shall name the same object wherever they occur).

If the name of an implementation-supplied function or the keyword TAB is implicitly or explicitly defined or declared as the identifier of a user-defined function, array, or variable, then the defined or declared interpretation of the identifier shall override the interpretation specified by the standard within the scope of the definition or declaration. Therefore, within that scope, the implementation-supplied function or the tab-call shall be unavailable.

Within any program-unit, identifiers that differ only in the cases of the letters they contain shall denote the same object (e.g., Xl identifies the same object as xl). Identifiers that differ in any other respect shall denote different objects.

4.4.5 Exceptions

None.

4.4.6 Remarks

No implementation-defined enhancement to this standard may extend the list of words unavailable for use as simple-variables. Since all arrays shall be declared (cf. 7.1), and since all

defined-functions shall be declared or defined in the program-unit in which they are referenced, implementations may supply built-in functions other than those specified in this standard provided that any declaration for such identifiers within a program overrides the implementation-supplied interpretation. Note, however, that in some cases the use of a parameterless function supplied by an implementation as an enhancement would be syntactically indistinguishable from a variable having the same name. Therefore, implementations that provide such functions shall also provide a syntactic means for identifying them as functions. Examples of such syntax are (1) a requirement to declare such functions explicitly in any program-unit where they are used, or (2) requiring the use of empty parentheses (e.g., "NEWFUNCTION()") with references to such functions.

An operating system may impose additional restrictions on the length and form of identifiers for procedures that are compiled independently of the main-program.

A supplied-function may be overridden by defining a user-defined function or simple-variable with the same name. An identifier may have the same spelling as a keyword (other than PRINT, ELSE, REM, or NOT).

5. Numbers

Numbers constitute one of two primitive data types in BASIC (the other is strings). Constants, variables, and implementation-supplied functions, which can be used to form expressions, are associated with numbers.

5.1 Numeric Constants

5.1.1 General Description

Numeric-constants denote scalar numeric values. A numeric-constant is a decimal representation, in positional notation, of a number. There are four general syntactic forms of numeric-constants:

(1) Implicit point unscaled representation sd...d
(2) Explicit point unscaled representation sd...drd...d
(3) Explicit point scaled representation sd...drd...dEsd...d
(4) Implicit point scaled representation sd...dEsd...d

where d is a digit, r is a period, s is an optional sign, and E is the explicit character E or e. A numeric-constant not preceded by a sign is assumed to be positive.

5.1.2 Syntax

1. constant > numeric-constant
2. numeric-constant = sign? numeric-rep
3. sign = plus-sign / minus-sign
4. numeric-rep = significand exrad?
5. significand = integer period? / integer? fraction
6. integer = digit digit*
7. fraction = period integer
8. exrad = E sign? integer

5.1.3 Examples

2. -21.
4. 1E10
   5e-1
   .4E+1
5. 500.
   1.2
7. .255

5.1.4 Semantics

The value of a numeric-constant is the number represented by that constant. "E" and "e" stand for "times ten to the power"; if no sign follows the symbols E or e, then a plus-sign is understood.

A program can contain numeric-constants that have an arbitrary number of digits. An implementation shall retain either the exact value of a numeric-constant, or that value rounded to an implementation-defined precision. The implementation-defined precision for numeric-constants shall be not less than ten or six significant decimal digits, depending upon whether the arithmetic option in force is DECIMAL or NATIVE respectively (cf. 5.6). Numeric-constants can also have an arbitrary number of digits in the exrad, though nonzero constants whose magnitude is outside an implementation-defined range may be treated as exceptions (cf. 5.6). Nonzero constants whose magnitudes are less than machine infinitesimal shall be be replaced by zero, while constants whose magnitudes are larger than MAXNUM shall be reported as causing an overflow.

5.1.5 Exceptions

The evaluation of a numeric-constant causes an overflow (1001, fatal).

5.1.6 Remarks

It is recommended that implementations report constants whose magnitudes are less than machine infinitesimal as underflows (1501, nonfatal: replace by zero and continue) to permit interception by exception handlers.

Although this standard contains no provision for named constants, their effect can be achieved through no-argument defined-functions (cf. 9.1).

5.2 Numeric Variables

5.2.1 General Description

Numeric-variables may be either simple-numeric-variables or references to elements of numeric-arrays.

5.2.2 Syntax

```
 1. variable                  > numeric-variable
 2. numeric-variable          = simple-numeric-variable /
                                numeric-array-element
 3. simple-numeric-variable   = numeric-identifier
 4. numeric-array-element     = numeric-array subscript-part
 5. numeric-array             = numeric-identifier
 6. subscript-part            = left-parenthesis subscript
                                (comma subscript)*
                                right-parenthesis
 7. subscript                 = index
 8. index                     = numeric-expression
 9. simple-variable           > simple-numeric-variable
10. array-name                > numeric-array
```

The number of subscripts in a subscript-part shall be one, two, or three.

5.2.3 Examples

```
3. X
   sum
4. V(4)
   table(i,j+1)
```

5.2.4 Semantics

At any instant in the execution of a program, a numeric-variable is associated with a single numeric value. The value associated with a numeric-variable may be changed by the execution of statements in the program.

Simple-numeric-variables are declared implicitly through their appearance in a program-unit. The scope of a numeric-variable shall be the program-unit in which it appears, unless it is a parameter of an internal-function-def (cf. 9.1).

An index is a numeric-expression whose value shall be rounded to the nearest integer; the rounded value of X is defined to be INT(X+.5).

A numeric-array-element is called a subscripted numeric-variable and refers to the element in the array selected by the value(s) of the subscript(s). The acceptable range of values shall be explicitly declared in a dimension-statement or a declare-statement (cf. 7.1). Subscripts shall have values within the appropriate range.

At the initiation of execution the values associated with all numeric-variables shall be implementation-defined.

5.2.5 Exceptions

A subscript is not in the range of the declared bounds (2001, fatal).

5.2.6 Remarks

Since initialization of variables is not specified, and hence may vary from implementation to implementation, programs that are intended to be transportable should explicitly assign a value to each variable before any expression involving that variable is evaluated.

There are many commonly used alternatives for associating implementation-defined initial values with variables; it is recommended that all variables be recognizably undefined in the sense that an exception will result from any attempt to access the value of any variable before that variable is explicitly assigned a value (3101, nonfatal: supply an implementation-defined value and continue).

5.3 Numeric Expressions

5.3.1 General Description

Numeric-expressions may be constructed from numeric-variables, numeric-reps, and numeric-function-refs using the operations of addition, subtraction, multiplication, division, and exponentiation (i.e., raising to a power).

5.3.2 Syntax

```
 1. expression            > numeric-expression
 2. numeric-expression    = sign? term (sign term)*
 3. term                  = factor (multiplier factor)*
 4. factor                = primary (circumflex-accent primary)*
 5. primary               = numeric-rep / numeric-variable /
                            numeric-function-ref /
                            left-parenthesis numeric-expression
                            right-parenthesis
 6. numeric-function-ref  > numeric-function function-arg-list?
 7. numeric-function      = numeric-defined-function /
                            numeric-supplied-function
 8. function-arg-list     = left-parenthesis function-argument
                            (comma function-argument)*
                            right-parenthesis
 9. function-argument     = expression / actual-array
10. actual-array          = array-name
11. multiplier            = asterisk / slant
```

The number and types of arguments in a numeric-function-ref shall agree with the number and types of corresponding parameters in the definition of the numeric-function. An actual-array shall have the same number of dimensions as the corresponding parameter.

Whenever numeric arguments are passed to an external-function-def, the ARITHMETIC options in effect for the external-function-def and the invoking program-unit shall agree.

Each numeric-function referenced in an expression within a program-unit shall either be implementation-supplied, or shall be defined in an internal-function-def or declared in a declare-statement occurring in a lower-numbered line, within the same program-unit, than the first reference to that numeric-function.

5.3.3 Examples

2. 3*X - Y^2
   cost*quantity + overhead
4. 2^(-X)
5. SQR(X^2+Y^2)
6. value(X,Y,a$)
   minimum(Xvector)

5.3.4 Semantics

The formation and evaluation of numeric-expressions follows the normal algebraic rules. The symbols circumflex-accent (^), asterisk (*), slant (/), plus-sign (+), and minus-sign (-) represent the operations of exponentiation, multiplication, division, addition, and subtraction or negation, respectively. Unless parentheses dictate otherwise, exponentiations shall be performed first, then multiplications and divisions, and finally additions, subtractions, and negations. In the absence of parentheses, operations of the same precedence shall be evaluated from left to right. Thus A-B-C shall be interpreted as (A-B)-C; A^B^C, as (A^B)^C; A/B/C, as (A/B)/C; -A+B as (-A)+B; and -A^B as -(A^B).

For those mathematical operators that are associative, commutative, or both, full use of these properties may be made in order to revise the order of evaluation of the numeric-expression, except where constrained by the use of parentheses.

If an underflow occurs in the evaluation of a numeric-expression, then the value generated by the operation that resulted in the underflow shall be replaced by zero.

0^0 is defined to be 1.

A numeric-function-ref is a notation for the invocation of a predefined algorithm, into which the argument values, if any, shall be substituted for the parameters (cf. 5.4, 6.4, and 9.1) used in the function-def. The result of evaluating a numeric-function, achieved by the execution of the defining algorithm, shall be a scalar numeric value, which replaces the numeric-function-ref in the numeric-expression.

5.3.5 Exceptions

Evaluation of a numeric-expression results in division by zero (3001, fatal).

Evaluation of a numeric-expression results in an overflow (1002, fatal).

Evaluation of the operation of exponentiation results in a negative number being raised to a non-integer power (3002, fatal).

Evaluation of the operation of exponentiation results in zero being raised to a negative power (3003, fatal).

5.3.6 Remarks

The accuracy with which the evaluation of a numeric-expression takes place may vary from implementation to implementation, subject to the constraints of 5.6.

It is recommended that implementations report underflow as an exception (1502, nonfatal: replace by zero and continue) to permit interception by exception handlers.

Implementations may evaluate primaries and operations within a numeric-expression in any order that is consistent with the semantics of 5.3.4. (Of course, an operation must be evaluated after its operands.) For example, in the expression "A+B+C+D*E", the primaries and additions may be evaluated in any order. However, the multiplication must be performed before the addition implied by the third plus-sign, since the product "D*E" is one of the operands of that addition.

5.4 Implementation-Supplied Numeric Functions

5.4.1 General Description

Predefined algorithms are supplied by the implementation for the evaluation of commonly used numeric functions. Additional functions related to other features of this standard are defined in 6.4, 7.1, 7.2, 12.1, 13.5, and 14.7.

5.4.2 Syntax

1. numeric-supplied-function > ABS / ACOS / ANGLE / ASIN / ATN / CEIL / COS / COSH / COT / CSC / DATE / DEG / EPS / EXP / FP / MAXNUM / INT / IP / LOG / LOG10 / LOG2 / MAX / MIN / MOD / PI / RAD / REMAINDER / RND / ROUND / SEC / SGN / SIN / SINH / SQR / TAN / TANH / TIME / TRUNCATE
2. randomize-statement = RANDOMIZE

5.4.3 Examples

2. RANDOMIZE

5.4.4 Semantics

The values of the numeric-supplied functions, as well as the number of arguments required for each function, shall be as described below. In all cases, X and Y stand for numeric-expressions, and N stands for an index, i.e., the rounded integer value of a numeric-expression. Each function accepts numeric arguments within the range of the negative number with the largest magnitude to the largest positive number, except where noted. For functions that return a value in angle measure (ACOS, ANGLE, ASIN, and ATN), the value shall be in radians unless OPTION ANGLE DEGREES is in effect (cf. 5.6), when the value shall be in degrees. In the semantics below, "pi" (lower-case) stands for the true value of that constant.

Function	Function Value
ABS(X)	The absolute value of X.

ACOS(X) The arccosine of X in radians or degrees (cf. 5.6), where $0 \leq ACOS(X) \leq pi$; X shall be in the range $-1 \leq X \leq 1$.

ANGLE(X,Y) The angle in radians or degrees (cf. 5.6) between the positive x-axis and the vector joining the origin to the point with coordinates (X,Y), where $-pi < ANGLE(X,Y) \leq pi$. X and Y shall not both be 0. Note that counterclockwise is positive (e.g., ANGLE(1,1) = 45 degrees).

ASIN(X) The arcsine of X in radians or degrees (cf. 5.6), where $-pi/2 \leq ASIN(X) \leq pi/2$; X shall be in the range $-1 \leq X \leq 1$.

ATN(X) The arctangent of X in radians or degrees (cf. 5.6), i.e., the angle whose tangent is X, where $-(pi/2) < ATN(X) < (pi/2)$.

CEIL(X) The smallest integer not less than X.

COS(X) The cosine of X, where X is in radians or degrees (cf. 5.6).

COSH(X) The hyperbolic cosine of X.

COT(X) The cotangent of X, where X is in radians or degrees (cf. 5.6).

CSC(X) The cosecant of X, where X is in radians or degrees (cf. 5.6).

DATE The current date in decimal form YYDDD, where YY are the last two digits of the year and DDD is the ordinal number of the current day of the year (e.g., the value of DATE on May 9, 1977 was 77129). If there is no calendar available, then the value of DATE shall be -1.

DEG(X) The number of degrees in X radians.

EPS(X) The maximum of (X-X', X"-X, sigma) where X' and X" are the predecessor and successor of X and sigma is the smallest positive value representable. If X has no predecessor, then X' is X; if X has no successor, then X" is X. Note EPS(0) is the smallest positive number representable by the implementation, and is therefore implementation-defined. Note also that

EPS may produce different results for different arithmetic options (cf. 5.6).

EXP(X) The exponential of X, i.e., the value of the base of natural logarithms (e = 2.71828...) raised to the power X; if EXP(X) is less than machine infinitesimal, then its value shall be replaced by zero.

FP(X) The fractional part of X, i.e., X - IP(X).

INT(X) The largest integer not greater than X; e.g., INT(1.3) = 1 and INT(-1.3) = -2.

IP(X) The integer part of X, i.e., SGN(X)*INT(ABS(X)).

LOG(X) The natural logarithm of X; X shall be greater than zero.

LOG10(X) The common logarithm of X; X shall be greater than zero.

LOG2(X) The base 2 logarithm of X; X shall be greater than zero.

MAX(X,Y) The larger (algebraically) of X and Y.

MAXNUM The largest finite positive number representable and manipulable by the implementation; implementation-defined. MAXNUM may represent different numbers for different arithmetic options (cf. 5.6).

MIN(X,Y) The smaller (algebraically) of X and Y.

MOD(X,Y) X modulo Y, i.e., X-Y*INT(X/Y). Y shall not equal zero.

PI The constant 3.14159..., which is the ratio of the circumference of a circle to its diameter.

RAD(X) The number of radians in X degrees.

REMAINDER(X,Y) The remainder function, i.e., X-Y*IP(X/Y). Y shall not equal zero.

RND The next pseudorandom number in an implementation-defined sequence of pseudorandom numbers uniformly distributed in the range $0 \leq RND < 1$.

ROUND(X,N) The value of X rounded to N decimal digits to the right of the decimal point (or -N digits to the left if N < 0); i.e., INT(X*10^N+.5)/10^N.

SEC(X) The secant of X, where X is in radians or degrees (cf. 5.6).

SGN(X) The sign of X: -1 if X < 0, 0 if X = 0, and +1 if X > 0.

SIN(X) The sine of X, where X is in radians or degrees (cf. 5.6).

SINH(X) The hyperbolic sine of X.

SQR(X) The nonnegative square root of X; X shall be nonnegative.

TAN(X) The tangent of X, where X is in radians or degrees (cf. 5.6).

TANH(X) The hyperbolic tangent of X.

TIME The time elapsed since the previous midnight, expressed in seconds; e.g., the value of TIME at 11:15 AM is 40500. If there is no clock available, then the value of TIME shall be -1. The value of TIME at midnight shall be zero (not 86400).

TRUNCATE(X,N) The value of X truncated to N decimal digits to the right of the decimal point (or -N digits to the left if N < 0); i.e., IP(X*10^N)/10^N.

If OPTION ANGLE DEGREES is in effect, the term "in radians or degrees" in the above list of function values shall mean degrees. If OPTION ANGLE RADIANS is in effect, the term "in radians or degrees" shall mean radians. The accuracy requirements (cf. 5.6.4) for the periodic trigonometric functions SIN, COS, TAN, SEC, CSC, COT are limited to providing full accuracy of m+1 decimal digits only for arguments in the range of -2*pi to 2*pi. Loss of accuracy outside this range is limited to the result of loss of precision in performing those range reductions on arguments necessary to compute values of these functions, i.e., "SIN (x)" may be evaluated as if it were written "SIN (MOD (x, 2*pi))" and similarly for the other functions.

If no randomize-statement is executed, then the RND function shall generate the same sequence of pseudo-random numbers each time a program is run. Execution of a randomize-statement shall override this implementation-supplied sequence of pseudorandom numbers, generating a new (and unpredictable) starting point for the list of pseudorandom numbers used subsequently by the RND function. The sequence of pseudorandom numbers shall be global to the entire program, not local to individual program-units.

5.4.5 Exceptions

The value of the argument of the LOG, LOG10, or LOG2 function is zero or negative (3004, fatal).

The value of the argument of the SQR function is negative (3005, fatal).

The magnitude of the value of a numeric-supplied-function is larger than MAXNUM or is mathematical infinity (1003, fatal).

The value of the second argument of the MOD or REMAINDER function is zero (3006, fatal).

The value of the argument of the ACOS or ASIN function is less than -1 or greater than 1 (3007, fatal).

An attempt is made to evaluate ANGLE(0,0) (3008, fatal).

5.4.6 Remarks

In the case of implementations that do not have access to a randomizing device such as a real-time clock, the randomize-statement may be implemented by means of an interaction with the user.

This standard requires that overflows be reported only for the final values of numeric-supplied-functions; exceptions that occur in the evaluation of these functions need not be reported, though implementations shall take appropriate actions in the event of such exceptions to insure the accuracy of the final values. When overflows are reported for the final values of numeric-supplied-functions, it is recommended that the name of the function generating the overflow be reported also.

It is recommended that, if the magnitude of the value of a numeric-supplied-function is nonzero, but less than machine infinitesimal, implementations report this as an underflow, set

the value to zero (1503, nonfatal: return zero and continue) to permit interception by exception handlers.

The time-zone used for DATE and TIME is implementation-defined.

It may not be possible, for reasons of overflow, to express the year in full format in DATE. When this full format is needed, the function DATE$ should be used.

5.5 Numeric Assignment Statements

5.5.1 General Description

A let-statement provides for the simultaneous assignment of the computed value of a numeric-expression to a list of numeric-variables.

5.5.2 Syntax

```
1. let-statement           > numeric-let-statement
2. numeric-let-statement   = LET numeric-variable-list
                             equals-sign numeric-expression
3. numeric-variable-list   = numeric-variable
                             (comma numeric-variable)*
```

5.5.3 Examples

```
2. LET P = 3.14159
   LET A(X,3) = SIN(X)*Y + 1
   LET A, Y(I), Z = I + 1
   LET T(I,J), I, J = I + J
```

5.5.4 Semantics

The subscripts, if any, of variables in the numeric-variable-list shall be evaluated in sequence from left to right. Next the numeric-expression on the right of the equals-sign shall be evaluated (cf. 5.3). Finally, the value of that numeric-expression, if necessary rounded to the nearest value that can be retained by the variable, shall be assigned to the numeric-variables in the numeric-variable-list in order from left to right.

5.5.5 Exceptions

None.

5.5.6 Remarks

Note that:

```
LET A = 1
LET A, B(A) = 2
```

is not equivalent to:

```
LET A = 1
LET A = 2
LET B(A) = 2
```

5.6 Numeric Arithmetic and Angle

5.6.1 General Description

Unless specified otherwise, the values of all numeric-variables shall behave logically as floating-point decimal numbers with an implementation-defined precision of at least ten decimal digits. By use of an option-statement, a program may choose to take advantage of a more efficient, but possibly less accurate, representation for numeric values.

Unless specified otherwise, the trigonometric functions (cf. 5.4) and the graphical transform-functions (cf. 13.4) require arguments or generate values in radian measure. By use of an option-statement, a program may change the angle measure of all such functions to degrees.

5.6.2 Syntax

1.	option-statement	= OPTION option-list
2.	option-list	= option (comma option)*
3.	option	> ARITHMETIC (DECIMAL / NATIVE) / ANGLE (DEGREES / RADIANS)
4.	declare-statement	= DECLARE type-declaration
5.	type-declaration	> numeric-type
6.	numeric-type	> NUMERIC numeric-declaration (comma numeric-declaration)*
7.	numeric-declaration	> simple-numeric-variable

An option-statement with an ARITHMETIC option, if present at all, shall occur in a lower-numbered line than any numeric-expression, or a dimension-statement or a declare-statement referencing a numeric-array or fixed-declaration in the same program-unit.

A program-unit shall contain at most one ARITHMETIC option.

An ANGLE option, if present at all, shall occur in a lower-numbered line than any reference to any numeric-supplied-function or transform-function in the same program-unit.

A program-unit shall contain at most one ANGLE option.

A declare-statement, if present at all, shall occur in a lower-numbered line than any reference to the variables declared therein.

5.6.3 Examples

1. OPTION ARITHMETIC DECIMAL, ANGLE DEGREES

5.6.4 Semantics

The ARITHMETIC option controls the logical behavior of numeric entities within the program-unit containing the option.

If OPTION ARITHMETIC DECIMAL is specified, or if no ARITHMETIC option is specified, then the values of the numeric-variables shall behave logically as decimal floating-point numbers, with an implementation-defined precision, say m, of at least ten significant decimal digits and with an implementation-defined range of at least 1E-38 to 1E+38.

The results of decimal computations can be described in terms of floating-point decimal intermediate results with at least m+1 decimal digits of precision (but may be implemented in some other equivalent fashion). The value of a numeric-variable shall be assumed to be exact. Numeric-constants shall be evaluated accurately to at least m decimal digits of precision. Numeric operations and functions shall also be evaluated accurately to at least m+1 decimal digits of precision with respect to the computed value of their operands and arguments (which may themselves be intermediate results). In all cases, the intermediate result of an evaluation shall be represented as a floating-point decimal number with at least m+1 decimal digits of precision; thus, when the true result can be expressed as a decimal number with m+1 significant digits, the computed result shall be exact. In no case shall the error for evaluation of an individual constant, operation, or function be greater than 5 in the (m+2)nd significant digit. Implementations are free to use any method of numeric evaluation that always yields results whose absolute error (with respect to the true result) is no greater than the absolute error of the results generated by the preceding specification.

If OPTION ARITHMETIC NATIVE is specified, then the values of numeric variables and constants shall be represented and manipulated in an implementation-defined fashion, with an implementation-defined precision of at least six decimal digits and with an implementation-defined range of at least 2E-38 to 1E+38. Decimal values need not be represented exactly, as long as the error is within the limits of this precision.

The ANGLE option controls the evaluation of the trigonometric functions within the program-unit containing the option.

If OPTION ANGLE RADIANS is specified, or if no ANGLE option is specified, then the numeric-supplied-functions COS, COT, CSC, SEC, SIN, and TAN, as well as the graphic transform-functions ROTATE and SHEAR, use arguments in radian measure, and the numeric-supplied-functions ACOS, ANGLE, ASIN, and ATN generate results in radian measure.

If OPTION ANGLE DEGREES is specified, then the numeric-supplied-functions COS, COT, CSC, SEC, SIN, and TAN, as well as the graphic transform-functions ROTATE and SHEAR, use arguments in degree measure, and the numeric-supplied-functions ACOS, ANGLE, ASIN, and ATN generate results in degree measure.

If the execution of a program reaches a line containing an option-statement, then it shall proceed to the next line with no further effect.

A simple-numeric-variable that appears in a numeric-type shall establish that variable as a simple-numeric-variable.

If execution reaches a line containing a declare-statement, it shall proceed to the next line with no further effect.

5.6.5 Exceptions

None.

5.6.6 Remarks

The representations chosen for numeric values when OPTION ARITHMETIC NATIVE is specified may be the same as that for OPTION ARITHMETIC DECIMAL.

No minimum accuracy is specified for the evaluation of numeric expressions and functions when OPTION ARITHMETIC NATIVE has been chosen. However, it is recommended that implementations maintain at least six decimal digits of precision.

The value 2E-38 is specified for the maximum value of the lower bound of positive numbers to allow an implementation employing the IEEE floating point binary arithmetic to be standard conforming.

6. Strings

Character strings constitute one of two primitive data types in BASIC (the other is numbers). Strings consist of arbitrary sequences of characters. Their lengths are variable, not fixed, although a maximum length for a string may be specified. With strings are associated constants, variables, and implementation-supplied functions, from which expressions can be formed.

6.1 String Constants

6.1.1 General Description

A string-constant is a character string of fixed length enclosed within quotation-marks. A quotation-mark itself may be included in a string-constant by representing it by two adjacent quotation-marks.

6.1.2 Syntax

1. constant > string-constant
2. string-constant = quoted-string
3. quoted-string = quotation-mark quoted-string-character* quotation-mark

The length of a string-constant (i.e., the number of quoted-string-characters contained between the quotation-marks) shall be limited only by the implementation-defined maximum number of characters preceding each end-of-line indicator (i.e., at least 132).

6.1.3 Examples

2. "XYZ"
   "1E10"
   "He said, ""Don't""."

6.1.4 Semantics

The value of a string-constant shall be the sequence of all quoted-string-characters between the initial and final quotation-marks. The double-quote, when appearing inside a quoted-string, shall denote a single quotation-mark. Spaces in string-constants, including trailing spaces, shall be significant. A string consisting only of two quotation-marks shall represent the null string. Upper-case- and lower-case-letters shall be distinct within string-constants.

6.1.5 Exceptions

None.

6.1.6 Remarks

The maximum length of a string-constant is constrained by the maximum length of a physical line. The maximum length of the constant would therefore be 3 less than that for the line, allowing for a continuation character ("&"), and the leading and trailing quotation-mark, e.g.:

```
100 LET A$ =  &
&"abc...unseen characters here...xyz"
```

As the maximum physical line length shall be at least 132, the maximum string-constant length shall be at least 129.

6.2 String Variables

6.2.1 General Description

String-variables may be either simple-string-variables or references to elements of one-, two-, or three-dimensional string-arrays.

Explicit declarations of simple-string-variables are not required. A dollar-sign serves to distinguish a string-variable from a numeric-variable.

6.2.2 Syntax

```
1. variable                > string-variable
2. string-variable         = (simple-string-variable /
                             string-array-element)
                             substring-qualifier?
3. simple-string-variable  = string-identifier
4. string-array-element    = string-array subscript-part
5. string-array            = string-identifier
6. substring-qualifier     = left-parenthesis index colon
                             index right-parenthesis
7. simple-variable         > simple-string-variable
8. array-name              > string-array
```

6.2.3 Examples

```
2. K$
   name$(X:Y)
   ITEM$(1,n)(z:z+5)
4. A$(4)
   table$(I,J)
```

6.2.4 Semantics

At any instant in the execution of a program, a string-variable is associated with a single string value. The value associated with a string-variable may be changed by the execution of statements in the program.

The length of the character string associated with a string-variable can vary during the execution of a program from a length of zero characters (signifying the null or empty string) to the maximum allowed for that string-variable (cf. 6.6.4).

Simple-string-variables may be declared explicitly (cf. 6.6) or may be declared implicitly through their appearance in a program-unit. The scope of a string-variable shall be the program-unit in which it appears, unless it is a parameter of an internal-proc-def, in which case its scope is that definition.

A string-array element is called a subscripted string-variable and refers to the element in the one-, two-, or three-dimensional array selected by the value(s) of the subscript(s). Subscripts shall have values within the appropriate range (cf. 7.1).

The substring-qualifier provides a means for specifying a portion of the value associated with a string-variable. A$(M:N) shall specify that substring of the value associated with A$ from its Mth through its Nth characters (M and N are indices). Characters in a string shall be numbered from the left starting with one. There are no exceptions associated with substring-qualifiers; if either M or N is not in the range from 1 to LEN(A$), then M shall be considered to be MAX(M,1) and N shall be considered to be MIN(N,LEN(A$)). If M > N, even after this adjustment, then A$(M:N) shall be the null string occurring before the Mth character of A$ if M $\leq$ LEN(A$) or the null string immediately following A$ if M > LEN(A$). For example, if A$ = "1234", then A$(1:1) = "1", A$(1:3) = "123", A$(0:3) = "123", A$(2:5) = "234", A$(3:2) is the null string preceding the third character of A$, and A$(5:7) is the null string following A$.

At the initiation of execution the values associated with all string-variables shall be implementation-defined.

6.2.5 Exceptions

A subscript is not in the range of the declared bounds (2001, fatal).

6.2.6 Remarks

Since initialization of variables is not specified, and hence may vary from implementation to implementation, programs that are intended to be transportable should explicitly assign a value to each variable before any expression involving that variable is evaluated.

There are many commonly used alternatives for associating implementation-defined initial values with variables; it is recommended that all variables be recognizably undefined in the sense that an exception will result from any attempt to access the value of any variable before that variable is explicitly assigned a value (3102, nonfatal: supply an implementation-defined value and continue).

6.3 String Expressions

6.3.1 General Description

String-expressions are composed of string-variables, string-constants, string-function-references, or a concatenation of these.

6.3.2 Syntax

```
1. expression              > string-expression
2. string-expression       = string-primary
                             (concatenation string-primary)*
3. string-primary          = string-constant /
                             string-variable /
                             string-function-ref /
                             left-parenthesis string-expression
                             right-parenthesis
4. string-function-ref     = string-function function-arg-list?
5. string-function         = string-defined-function /
                             string-supplied-function
6. concatenation           = ampersand
```

The number and types of arguments in a string-function-ref shall agree with the number and types of the corresponding parameters specified in the definition of the string-function. An actual-array shall have the same number of dimensions as the corresponding parameter.

Each string-function referenced in an expression within a program-unit shall either be implementation-supplied, or shall be defined in an internal-function-def or declared in a declare-statement occurring in a lower-numbered line, within the same program-unit, than the first reference to that string-function.

6.3.3 Examples

```
2. A2$ & B$(4:22) & "223"
3. X$(1,3)(I:J)
```

6.3.4 Semantics

The value of a string-expression shall be the concatenation of the values of the string-primaries in the expression (e.g., if A$ = "COME " and B$ = "IN", then A$ & B$ = "COME IN" and B$ & A$ = "INCOME ").

Within a string-expression, string-primaries shall be evaluated from left to right. For each string-primary, the subscripts, if any, shall be evaluated first, then the substring-qualifiers, and then the value of the primary itself.

A string-function-ref is a notation for the invocation of a predefined algorithm, into which the argument values, if any, shall be substituted for the parameters (cf. 6.4 and 9.1) used in the function-def. The result of evaluating a string-function, achieved by the execution of the defining algorithm, shall be a scalar string value which replaces the string-function-ref in the string-expression.

6.3.5 Exceptions

Evaluation of a string-expression causes a string overflow (1051, fatal).

6.3.6 Remarks

The ampersand is used both for concatenation and line-continuation. Thus:

```
100  PRINT "ABC"  &&
&  "XYZ"
```

will print the sequence of characters ABCXYZ.

6.4 Implementation-Supplied String Functions

6.4.1 General Description

Predefined algorithms are supplied by the implementation for the evaluation of commonly used string-valued functions and numeric-valued functions whose arguments are strings.

6.4.2 Syntax

1. string-supplied-function > (CHR / DATE / LCASE / LTRIM / REPEAT / RTRIM / STR / TIME / UCASE / USING) dollar-sign
2. numeric-supplied-function > LEN / ORD / POS / VAL
3. numeric-function-ref > MAXLEN left-parenthesis (simple-string-variable / string-array) right-parenthesis

6.4.3 Examples

None.

6.4.4 Semantics

The values of the implementation-supplied functions, as well as the number and types of arguments required for each function, are described below. In all cases, M represents an index, i.e., the rounded integer value of some numeric-expression; X stands for a numeric-expression; V$ represents a simple-string-variable or string-array; and A$ and B$ stand for string-expressions.

Function	Function value
CHR$(M)	The one-character string consisting of the character occupying ordinal position M+1 in the collating sequence for the declared character set, i.e., the first character is returned for an argument of zero. M shall be at least zero and less than the number of characters in the declared character set (cf. Table 8). For example, for the standard character set, CHR$(53) = "5", and CHR$(65) = "A". The values of CHR$ for the native character set are implementation-defined.
DATE$	The date in the string representation "YYYYMMDD" according to ANSI X3.30-1985. For example, the value of DATE$ on May 9, 1977 was "19770509".

If there is no calendar available, then the value of DATE$ shall be "00000000".

LCASE$(A$) The string of characters resulting from the value associated with A$ by replacing each uppercase letter in the string by its lowercase version.

LEN(A$) The number of characters in the value associated with A$. Note that LEN("""") = 1, since the value of the string constant consists of precisely one quotation-mark.

LTRIM$(A$) The string of characters resulting from the value associated with A$ by deleting all leading space characters.

MAXLEN(V$) The maximum length associated with the simple-string-variable or string-array (cf. 6.6). If there is no effective limit on string length, the value returned shall be MAXNUM.

ORD(A$) The ordinal position of the character named by the string associated with A$ in the collating sequence of the declared character set, where the first member of the character set is in ordinal position zero. The acceptable values of A$ are single characters in the character set and two- or three-character mnemonics for characters in the character set. Values of A$ with two or more characters shall be treated with upper-case- and lower-case-letters equivalent. The acceptable values for the standard character set are shown in Table 8. The acceptable values for the native character set are implementation-defined. For example, for the standard character set, ORD("BS") = 8, ORD("A") = 65, ORD("a") = 97, ORD("5") = 53, ORD("SOH") = 1, ORD("Soh") = 1, and ORD("ABC") causes an exception.

POS(A$,B$) The character position, within the value associated with A$, of the first character of the first occurrence of the value associated with B$. If there is no such occurrence, then POS(A$,B$) shall be zero. POS(A$,"") shall be one, for all values of A$.

POS(A$,B$,M) The character position, within the value associated with A$, of the first character of the

first occurrence of the value associated with B$, starting at the Mth character of A$. If the value associated with B$ does not occur within the designated portion of the value associated with A$, or if M is greater than LEN(A$), the value returned is zero. Otherwise, the value returned is equivalent to

```
LET temp1  = MAX(1, MIN(M, LEN(A$) + 1))
LET temp2$ = A$(temp1: LEN(A$))
LET temp3  = POS(temp2$, B$)
IF temp3 = 0 THEN
   LET POS = 0
ELSE
   LET POS = temp3 + temp1 - 1
END IF
```

For example, if A$ has the value "GRANDSTANDING", then POS(A$,"AN",1) = 3, POS(A$,"AN",4) = 8, and POS(A$,"AN",9) = 0. POS(A$,"",M) shall be MAX(M,1), as long as M $\leq$ LEN(A$).

REPEAT$(A$,M) The string consisting of M copies of A$; M $\geq$ 0.

RTRIM$(A$) The string of characters resulting from the value associated with A$ by deleting all trailing space characters.

STR$(X) The string generated by the print-statement as the numeric-representation of the value associated with X. No leading or trailing spaces shall be included in this numeric-representation. For example, STR$(123.5) = "123.5" and STR$(-3.14) = "-3.14".

TIME$ The time of day in 24-hour notation according to ANSI X3.43-1986 (HH:MM:SS). For example, the value of TIME$ at 11:15 AM is "11:15:00". If there is no clock available, then the value of TIME$ shall be "99:99:99". The value of TIME$ at midnight is "00:00:00".

UCASE$(A$) The string of characters resulting from the value associated with A$ by replacing each lowercase letter in the string by its uppercase version.

USING$(A$,X) The string consisting of the formatted representation of X, using A$ as a format-item, according to the semantics of 10.4. The exceptions defined in 10.4.5 for formatted output also apply to the

USING$ function.

VAL(A$) The value of the numeric-constant associated with A$, if the string associated with A$ is a numeric-constant. Leading and trailing spaces in the string are ignored. If the evaluation of the numeric-constant would result in a value that causes an underflow, then the value returned shall be zero. For example, VAL(" 123.5 ") = 123.5, VAL("2.E-99") could be zero, and VAL("MCMXVII") causes an exception.

6.4.5 Exceptions

The value of the argument of VAL is not a valid numeric-constant (4001, fatal).

The value of the argument of VAL is a valid numeric-constant, but evaluating this constant results in an overflow (1004, fatal).

The value of the argument of CHR$ is not in the appropriate range (4002, fatal).

The value of the argument of ORD is neither a valid single character nor a valid mnemonic (4003, fatal).

The value of the second argument of REPEAT$ is not $\geq$ 0 (4010, fatal).

6.4.6 Remarks

It is recommended that if the magnitude of the value of the VAL function is less than machine infinitesimal, implementations report this as an exception (1504, nonfatal: replace with zero and continue) to permit interception by exception handlers.

The time zone used for DATE$ and TIME$ is implementation-defined.

The effect of the functions UCASE$ and LCASE$ is fully defined only for the ASCII character set as defined in 4.1.4. For other-characters, such as accented letters, the effect is implementation-defined, and may be specified in other national versions of this standard to accommodate the needs of local alphabets.

6.5 String Assignment Statements

6.5.1 General Description

A let-statement provides for the simultaneous assignment of the computed value of a string-expression to a list of string-variables.

6.5.2 Syntax

```
1. let-statement          > string-let-statement
2. string-let-statement   = LET string-variable-list
                            equals-sign string-expression
3. string-variable-list   = string-variable
                            (comma string-variable)*
```

6.5.3 Examples

```
2. LET A$ = "ABC"
   LET A$(I) = B$(3:4)
   LET A$, B$ = "NEGATIVE DISCRIMINANT"
   LET C$(7:10) = "wxyz"
   LET A$ = "ABCD" &&
   & "XYZ"
```

6.5.4 Semantics

The subscripts and substring-qualifiers, if any, of variables in the string-variable-list shall be evaluated in sequence from left to right. Next the string-expression on the right of the equals-sign shall be evaluated (cf. 6.3). Finally, the value of that string-expression shall be assigned to the string-variables in the string-variable-list in order from left to right.

When a value is assigned to a string-variable with a substring-qualifier, it shall replace the substring of the value of the string-variable specified by the substring-qualifier. The length of the value of the string-variable may change as a result of this replacement. For example, if A$ = "1234", then assigning "32" to A$(2:3) results in "1324", assigning "" to A$(2:3) results in "14", assigning A$(1:2) to A$(2:3) results in "1124", and assigning "5" to A$(2:1) results in "15234".

6.5.5 Exceptions

The assignment of a value to a string-variable causes a string overflow (1106, fatal).

6.5.6 Remarks

The order of assignment of values to string-variables in the string-variable-list is important in statements such as

```
LET A$(1:2), A$(2:3) = "X"
```

where different orders of assignment may produce different results.

6.6 String Declarations

6.6.1 General Description

An option-statement may be used to define an ordering on the set of all string characters.

A declare-statement may be used to set a maximum length for specified string-variables in a program-unit.

6.6.2 Syntax

```
1. option                      > COLLATE (NATIVE / STANDARD)
2. type-declaration            > string-type
3. string-type                 = STRING length-max?
                                   string-declaration
                                   (comma string-declaration)*
4. length-max                  = asterisk integer
5. string-declaration          > simple-string-declaration
6. simple-string-declaration   = simple-string-variable
                                   length-max?
```

An option-statement with a COLLATE option, if present at all, shall occur in a lower-numbered line than any string-expression, or a dimension-statement or declare-statement referencing a string-array or string-variable within the same program-unit. A program-unit shall contain at most one COLLATE option.

No simple-string-variable shall be declared more than once in a program-unit. A simple-string-variable that is a formal-parameter or a parameter shall not occur in a declare-statement.

6.6.3 Examples

```
1. COLLATE NATIVE
3. STRING*8 last_name$*20, first_name$, middle_name$
```

6.6.4 Semantics

The COLLATE option identifies the collating sequence to be used within a program-unit for comparing strings (cf. 8.1) and for computing values of the CHR$ and ORD functions (cf. 6.4). OPTION COLLATE NATIVE specifies that the native collating sequence of the host system shall be used. OPTION COLLATE STANDARD specifies that the collating sequence shall correspond to the order of the characters in Table 8. If no COLLATE option

appears in a program-unit, then the STANDARD collating sequence shall be used within that program-unit.

Simple-string-variables whose string-identifiers appear in string-types may have a maximum length less than or equal to the implementation-defined default value. The maximum is determined, in descending order of precedence, from (1) the length-max in the string-declaration for that variable, (2) the length-max in the string-type of the declare-statement containing that variable, or (3) the implementation-defined default. The length-max guarantees that string values up to that length may be stored in the variable and that an attempt to store a longer value will cause a string overflow exception. The implementation-defined maximum string length default shall be at least 132 characters.

A length-max of 0 in a string-type shall establish the associated string-variable as having a maximum length of 0 (i.e., the null string).

6.6.5 Exceptions

None.

6.6.6 Remarks

The native collating sequence may be the standard collating sequence.

The COLLATE option may be extended, on other national versions of this standard, to accommodate specific needs of local alphabets.

7. Arrays

Arrays are indexed collections of numbers or strings. Array elements can be manipulated by scalar numeric and string operations (cf. Sections 5 and 6). In addition, entire arrays may be manipulated by matrix statements.

7.1 Array Declarations

7.1.1 General Description

An option in the option-statement may be used to define the lower bound for all array subscripts within a program-unit that are not explicitly stated. By use of an option-statement the subscripts of all such arrays may be declared to have a lower bound of zero or one; if no such declaration occurs, the lower bound shall be one.

Arrays may have one, two, or three dimensions. The number of dimensions and subscript bounds for each dimension are declared in the declare-statement or dimension-statement. All array-names, except those appearing in a function-parm-list or a procedure-parm-list, shall be declared in one and only one such statement. If not explicitly declared, the lower subscript bound for a given dimension is one or zero, depending on the BASE option. Upper bounds shall always be explicitly declared.

A one-dimensional array with subscripts 1 to 10 or 1980 to 1989 or -9 to 0 contains 10 elements. A two-dimensional array with subscript bounds 1 to 10 for each dimension contains 100 elements. Similarly, a three-dimensional array with subscript-bounds 1 to 10 for each dimension contains 1000 elements.

A declare-statement can be used to dimension numeric-arrays as well as to declare maximum lengths for string-variables and string-arrays, and to dimension string-arrays. A dimension-statement can be used to dimension arrays, but not to declare the maximum length of strings in string-arrays.

7.1.2 Syntax

```
1. dimension-statement        = DIM dimension-list
2. dimension-list             = array-declaration
                                (comma array-declaration)*
3. array-declaration          = numeric-array-declaration /
                                string-array-declaration
4. numeric-array-declaration = numeric-array bounds
```

```
 5. bounds                    = left-parenthesis bounds-range
                                (comma bounds-range)*
                                right-parenthesis
 6. bounds-range              = signed-integer TO
                                signed-integer / signed-integer
 7. signed-integer            = sign? integer
 8. string-array-declaration  = string-array bounds
 9. option                    > BASE ( 0 / 1 )
10. string-declaration        > string-array-declaration
                                length-max?
11. numeric-declaration       > numeric-array-declaration
12. numeric-function-ref      > MAXSIZE maxsize-argument /
                                SIZE bound-argument /
                                LBOUND bound-argument /
                                UBOUND bound-argument
13. maxsize-argument          = left-parenthesis actual-array
                                right-parenthesis
14. bound-argument            = left-parenthesis actual-array
                                (comma index)? right-parenthesis
```

The number of bounds-ranges in a bounds shall be one, two, or three.

An array that is named as a formal-array of a defined-function, a subprogram, a program, or a picture-def shall not be declared in a declare-statement or dimension-statement (since the formal-array in the function- or procedure-parm-list serves as its declaration). Any other array shall be so declared in a lower numbered line than any reference to that array or one of its elements. Any reference to an array and its elements shall agree in dimensionality with the declaration of that array in a declare-statement, a dimension-statement, or as a function- or procedure-parameter.

No numeric- or string-array shall be dimensioned or declared more than once in a program-unit.

If the optional lower bound (the first signed-integer) is included in the bounds-range, it shall be less than or equal to the upper bound (the second signed-integer).

If the lower bound is not specified, then the upper bound shall not be less than the default lower bound, which may be zero or one, depending on the BASE option.

An option-statement with a BASE option, if present at all, shall occur in a lower-numbered line than any declare-statement or dimension-statement or any MAT statement that uses a numeric-

array-value in the same program-unit. A program-unit shall contain at most one BASE option.

If a bound-argument does not specify an index, the actual-array shall be declared as one-dimensional.

7.1.3 Examples

```
 1. DIM A(6), B(10,10), B$(100), D(1 TO 5, 1980 TO 1989)
    DIM A$(4,4), C(-5 TO 10)
10. A$(3 TO 21) * 8
12. SIZE(A,1)
    SIZE(B$,2)
    SIZE(X)
    LBOUND(A)
    UBOUND(C$,2)
```

7.1.4 Semantics

Each array-declaration declares the named array named to be either one-, two-, or three-dimensional, according to whether one, two, or three bounds-ranges are specified in the bounds for the array. In addition, the bounds specify the maximum and optionally minimum values that subscripts for the array shall have. If a minimum subscript is not explicitly declared and no BASE option occurs within the program-unit, then it shall be implicitly declared to be one.

The BASE option in an option-statement is local to the program-unit in which it occurs and declares the minimum value for all array subscripts in that program-unit that are not explicitly declared.

If the execution of a program reaches a line containing a dimension-statement, then it shall proceed to the next line with no further effect.

String-array-declarations appearing in a string-declaration may include a length-max, which sets the maximum length of each element of the string-array. As with simple-string-variables, if there is no length-max in the string-declaration, then the length-max, if any, of the string-type shall take effect. If there is no length-max in either, then the implementation-defined length-max, if any, shall take effect.

The value of SIZE(A,N) in which A is an actual-array and N is an index shall be the current number of permissible values for the Nth subscript of the array named by A (the value of N is

rounded to the nearest integer, and the subscripts of A are indexed from left to right, starting at one). The value of SIZE(A) shall be the current number of elements in the entire array A.

The value of MAXSIZE(A) shall be the total number of elements of the entire array named by A permitted by the array-declaration.

The value of LBOUND(A,N), where A is an actual-array and N is an index, shall be the current minimum value allowed for the Nth subscript of the array named by A. The value of UBOUND(A,N) shall be the current maximum value allowed for the Nth subscript of array A. As in the SIZE function, the value of N is rounded to the nearest integer, and the subscripts of array A are indexed from left to right, starting at one. The LBOUND and UBOUND functions may be called with a single argument, provided that argument is a vector, in which case the values of LBOUND and UBOUND are the current minimum and maximum values allowed for the subscript of the vector. (Here, and in the rest of Section 7, the word "vector" shall mean a "one-dimensional array" and the word "matrix" shall mean a "two-dimensional array".

7.1.5 Exceptions

The value of the index in a SIZE reference is less than one or greater than the number of dimensions in the array (4004, fatal).

The value of the index in an LBOUND reference is less than one or greater than the number of dimensions in the array (4008, fatal).

The value of the index in a UBOUND reference is less than one or greater than the number of dimensions in the array (4009, fatal).

7.1.6 Remarks

The dimension statement is retained for compatibility with minimal BASIC. All its capabilities are included within the declare-statement.

If an implementation supports more than three dimensions, SIZE, LBOUND, and UBOUND should work for those extra dimensions, and an exception should be generated only when an attempt is made to inquire about a dimension beyond those declared.

7.2 Numeric Arrays

7.2.1 General Description

Numeric-arrays in BASIC may be manipulated element-by-element. However, it is often more convenient to regard numeric-arrays as entities rather than as indexed collections of entities, and to manipulate the entire entity at once. BASIC provides a number of standard operations to facilitate such manipulations.

7.2.2 Syntax

```
 1. array-assignment           > numeric-array-assignment
 2. numeric-array-assignment   = MAT numeric-array equals-sign
                                 numeric-array-expression
 3. numeric-array-expression   = (numeric-array
                                 numeric-array-operator)?
                                 numeric-array /
                                 scalar-multiplier
                                 numeric-array /
                                 numeric-array-value /
                                 numeric-array-function-ref
 4. numeric-array-operator     = sign / asterisk
 5. scalar-multiplier          = primary asterisk
 6. numeric-array-value        > scalar-multiplier?
                                 (CON / IDN / ZER) redim?
 7. redim                      = left-parenthesis redim-bounds
                                 (comma redim-bounds)*
                                 right-parenthesis
 8. redim-bounds               = (index TO)? index
 9. numeric-array-function-ref = (TRN / INV) left-parenthesis
                                 numeric-array
                                 right-parenthesis
10. numeric-function-ref       > DET (left-parenthesis
                                 numeric-array
                                 right-parenthesis) /
                                 DOT left-parenthesis
                                 numeric-array comma
                                 numeric-array
                                 right-parenthesis
```

The number of redim-bounds in a redim shall be one, two, or three.

A numeric-array being assigned a value by a numeric-array-assignment shall have the same number of dimensions as the value of the numeric-array-expression.

The numeric-arrays in a numeric-function-ref involving DOT shall be one-dimensional.

There shall be no more than two redim-bounds following IDN.

The numeric-arrays in a sum or difference shall have the same number of dimensions. The numeric-array serving as the argument of DET, INV, or TRN shall be two-dimensional.

The numeric-arrays serving as operands for the numeric-array-operator asterisk (matrix multiply) shall be either one-dimensional or two-dimensional, and at least one of them shall be two-dimensional.

7.2.3 Examples

In the following examples A, B, and C are doubly-subscripted numeric-arrays, X, Y, and Z are singly-subscripted numeric-arrays, and W is a numeric-expression.

```
  2. MAT A = B              MAT X = Y
     MAT A = B + C          MAT X = Y - Z
     MAT A = B*C            MAT X = A*Y            MAT X = Y*A
     MAT A = W * B          MAT X = W * CON
     MAT A = ZER(4,3)       MAT X = ZER
     MAT A = INV(B)         MAT A = TRN(B)
 10. DET(B)                 DOT(X,Y)
```

7.2.4 Semantics

7.2.4.1 Array Assignments and Redimensioning. Execution of a numeric-array-assignment shall cause the numeric-array-expression to be evaluated and its value assigned to the array named to the left of the equals-sign. If necessary, this array shall have its size changed dynamically (i.e., its number of dimensions shall be unchanged, but its size in each dimension shall be changed to conform to the size of the array given by the value of the numeric-array-expression).

When the size of a numeric-array is changed dynamically, the current upper bounds for its subscripts shall be changed to conform to the new sizes. That is,

new_lower_bound = old_lower_bound
new_upper_bound = old_lower_bound + new_size - 1

The new sizes need not individually be less than or equal to the sizes determined in the array-declaration for that numeric-array, as long as the new total number of elements for the numeric-array

does not exceed the total number of elements determined by the array-declaration for that array.

7.2.4.2 Array expressions. The evaluation of numeric-array-expressions shall follow the normal rules of matrix algebra. The symbols asterisk "*", plus "+", and minus "-" shall represent the operations of multiplication, addition, and subtraction, respectively.

The dimensions of numeric-arrays in numeric-array-expressions shall conform to the rules of matrix algebra. The numeric-arrays in a sum or difference shall have the same sizes in each dimension. The numeric-arrays in a product shall have sizes L x M and M x N for some L, M, and N (in which case the product shall have size L x N), or an M element vector and a size M x N matrix (in which case the product shall be an N element vector), or a size L x M matrix and an M element vector (in which case the product shall be an L element vector). All elements in a numeric-array shall be used when evaluating a numeric-array-expression; i.e., each numeric-array shall be treated as an entity.

When a scalar-multiplier is present in a numeric-array-expression, the primary shall be evaluated, and then each element of the numeric-array shall be multiplied by this value.

If an underflow occurs in the evaluation of a numeric-array-expression, then the value generated by the operation that resulted in the underflow shall be replaced by zero.

7.2.4.3 Array values. Numeric-array-values shall be assigned to the numeric-array on the left of the equals sign. If no redim is present, the size of the numeric-array generated shall be the same as the size of the numeric-array to which it is to be assigned. If a redim is present, a numeric-array of the dimensions specified shall be generated, and the numeric-array to which it is assigned shall be redimensioned as described in 7.2.4.1. In a redim-bounds, the values of the indices are the lower and upper bounds of the corresponding dimension in the associated array-value. If the redim-bounds consists of a single index, its value shall be the upper bound, and the lower bound shall be the current default lower bound in effect. If a redim is used with the IDN constant, then it shall produce a square matrix; i.e., the number of rows shall equal the number of columns. If a redim is not used with the IDN constant, the numeric-array being assigned to shall be square.

The ZER constant shall generate a numeric-array, all of whose elements are zero. The CON constant shall generate a numeric-array, all of whose elements are one. The IDN constant shall generate an identity matrix, i.e., a square matrix with ones on the main diagonal and zeroes elsewhere. If only one redim-bounds is used with IDN, then the effect is just as if that redim-bounds had been specified twice.

If a scalar-multiplier is used with an IDN, ZER, or CON constant, then the primary (see 5.3) is evaluated and each nonzero element of the IDN, ZER, or CON constant is replaced by the value of the primary.

7.2.4.4 Array functions. The function TRN shall produce the transpose of its argument. An N x M matrix is returned for an M x N argument.

The function INV shall produce the inverse of its argument. The argument shall be a square matrix.

The function DET shall return the determinant of its argument. The argument shall be a square matrix.

The value of DOT(X,Y) shall result in a scalar value, which is the result of the inner product multiplication of the one-dimensional numeric-vectors X and Y.

7.2.5 Exceptions

The sizes of numeric-arrays in a numeric-array-expression do not conform to the rules of matrix algebra (6001, fatal).

The total number of elements required for a redimensioned array exceeds the number of elements reserved by the array's original dimensions (5001, fatal).

The first index in a redim-bounds is greater than the second (6005, fatal).

A redim-bounds consists of a single index that is less than the default lower bound in effect (6005, fatal).

The redim following IDN does not specify a square matrix, or no redim is present and the receiving matrix is not square (6004, fatal).

The argument of the DET function is not a square numeric matrix (6002, fatal).

The argument of the INV function is not a square numeric matrix (6003, fatal).

Evaluation of a numeric-array-expression results in an overflow (1005, fatal).

Evaluation of DET or DOT results in an overflow (1009, fatal).

Application of INV to a singular matrix, or loss of all significant digits (3009, fatal).

7.2.6 Remarks

It is recommended that implementations report underflow as an exception (1505, nonfatal: replace by zero and continue) to permit interception by exception handlers.

7.3 String Arrays

7.3.1 General Description

As with numeric-arrays, string-arrays may be regarded as entities rather than as indexed collections of entities. BASIC provides the ability to concatenate and assign entire arrays of strings.

7.3.2 Syntax

```
1. array-assignment              > string-array-assignment
2. string-array-assignment       = MAT string-array
                                   substring-qualifier?
                                   equals-sign
                                   string-array-expression
3. string-array-expression       = string-array-primary
                                   (concatenation
                                   string-array-primary)? /
                                   string-primary
                                   concatenation
                                   string-array-primary /
                                   string-array-primary
                                   concatenation
                                   string-primary /
                                   string-array-value
4. string-array-primary          = string-array
                                   substring-qualifier?
5. string-array-value            = (string-primary
                                   concatenation)?
                                   NUL dollar-sign redim?
```

A string-array being assigned a value by a string-array-assignment shall have the same number of dimensions as the value of the string-array-expression.

Two string-arrays being concatenated shall have the same number of dimensions.

7.3.3 Examples

```
2. MAT A$ = A$ & B$
   MAT A$ = NUL$(5,6)
   MAT A$ = ("Number") & B$
   MAT A$(4:6) = (" ") & B$
```

7.3.4 Semantics

Execution of a string-array-assignment shall cause the string-array-expression to be evaluated and its value assigned to the array named to the left of the equals-sign. If appropriate, this array shall have its size changed dynamically (i.e., its number of dimensions shall be unchanged, but its size in each dimension shall be changed to conform to the size of the array given by the value of the string-array-expression).

When the size of a string-array is changed dynamically, the current upper bounds for its subscripts shall be changed to conform to the new sizes. That is,

```
new_lower_bound = old_lower_bound
new_upper_bound = old_lower_bound + new_size - 1
```

The new sizes need not individually be less than or equal to the sizes determined in the string-array-declaration for that string-array, as long as the new total number of elements for the string-array does not exceed the total number of elements determined by the array-declaration for that array.

When a string-array on the left of a string-array-assignment has a substring-qualifier, the assignment to each element of the string-array shall replace the substring of the value of each element specified by the substring-qualifier. The substring-qualifier on the left shall be evaluated before the string-array-expression.

String-array-expressions involve the operations of concatenation and substring extraction. Two string-arrays being concatenated shall have the same size in each dimension; the concatenation shall be performed element-by-element. When concatenation is by a scalar, this scalar shall be prefixed or suffixed, as appropriate, to every element of the string-array. When a substring-qualifier is applied to a string-array, then the specified substring shall be extracted from each element in the array.

The order of evaluation and assignment shall be as follows:

(1) Evaluate the substring-qualifiers in the string-array on the left.
(2) Evaluate the string-array-expression from left to right, by evaluating each string-primary or string-array-primary as follows: evaluate first the subscripts, if any, then the substring qualifiers, and then the value of the primary itself.
(3) Concatenate.
(4) Make the assignment.

The string-array-value NUL$ is an array all of whose elements are the null string. If a redim is not present, the size of the string-array generated shall be the same as the size of the string-array to which it is to be assigned. If a redim is present, a string-array of the dimensions specified shall be generated and the string-array to which it is assigned shall be redimensioned as described above. The rules in 7.2.4 for redims with numeric-array-values apply to NUL$ as well.

7.3.5 Exceptions

The arrays in a string-array-expression have different sizes (6101, fatal).

The first index in a redim-bounds is greater than the second (6005, fatal).

A redim-bounds consists of a single index that is less than the default lower bound in effect (6005, fatal).

The total number of elements required for a redimensioned array exceeds the number of elements reserved by the array's original dimensions (5001, fatal).

Evaluation of a string-array-expression results in a string overflow (1052, fatal).

Assignment of a value to a string-array causes a string overflow (1106, fatal).

7.3.6 Remarks

None.

8. Control Structures

Control structures govern the order of execution of lines in a program, both by statements that make explicit reference to line-numbers and also by explicitly-constructed loops and decision mechanisms, which make no reference to line-numbers.

8.1 Relational Expressions

8.1.1 General Description

Relational-expressions enable the values of expressions to be compared in order to influence the flow of control in a program.

8.1.2 Syntax

```
 1. relational-expression  = disjunction
 2. disjunction            = conjunction (OR conjunction)*
 3. conjunction            = relational-term
                             (AND relational-term)*
 4. relational-term        = NOT? relational-primary
 5. relational-primary     = comparison / left-parenthesis
                             relational-expression
                             right-parenthesis
 6. comparison             = numeric-expression relation
                             numeric-expression /
                             string-expression relation
                             string-expression
 7. relation               = equality-relation /
                             greater-than-sign /
                             less-than-sign / not-greater /
                             not-less
 8. equality-relation      = equals-sign / not-equals
 9. not-equals             = less-than-sign greater-than-sign /
                             greater-than-sign less-than-sign
10. not-less               = greater-than-sign equals-sign /
                             equals-sign greater-than-sign
11. not-greater            = less-than-sign equals-sign /
                             equals-sign less-than-sign
```

8.1.3 Examples

```
 2. NOT X < Y OR A$ = B$ AND B$ = C$
 3. A <= X AND X <= B
    1 <= I AND I <= 10 AND A(I) = X
    I < N AND (J > M OR A(I) < B(J))
```

8.1.4 Semantics

The relation "less than or equal to" is denoted by not-greater. The relation "greater than or equal to" is denoted by not-less. The relation "not equal to" is denoted by not-equals. The relations "greater than," "less than," and "equals" are denoted by the corresponding syntactic sign.

The relation of equality shall hold between two numeric-expressions if and only if the two numeric-expressions have the same value.

The relation of equality shall hold between two string-expressions if and only if the values of the two string-expressions have the same length and contain identical sequences of characters.

In the evaluation of relational-expressions involving string-expressions, the relation "less than" shall be interpreted to mean "earlier in the collating sequence than", and the other relations shall be defined in a corresponding manner. More precisely, if two unequal strings in a relational-expression have the same length, then one shall be "less than" the other if, in the leftmost character position in which they differ, the character in that string precedes the character in the other according to the established collating sequence (cf. 6.6). If the two strings in a relational-expression have different lengths and one has zero length or is an initial leftmost segment of the other, then the shorter string shall be "less than" the other. Otherwise, the relationship between two strings of unequal length shall be determined by the contents of the shorter string and the leftmost portion of the longer string that is of the same length as the shorter string.

The precedence of the operators AND, OR, and NOT shall be as implied by the formal syntax. That is, NOT operates only on the relational-primary immediately following it, AND applies to the relational-terms immediately preceding and following it, and OR applies to the conjunctions immediately preceding and following it.

The order of evaluation of relational-expressions shall be as follows. The relational-expression shall take on the truth-value of the disjunction that constitutes it. The conjunctions immediately contained in the disjunction shall be evaluated from left to right until a true conjunction is found or none are left. As soon as a true conjunction is found, the whole disjunction is evaluated as true, and any remaining conjunctions are not

evaluated. If no true conjunctions are found, the disjunction is false. For each conjunction, the relational-terms immediately contained in it are evaluated from left to right until a false relational-term is found or none are left. As soon as a false relational-term is found, the whole conjunction is evaluated as false and any remaining relational-terms are not evaluated. If all the relational-terms are true, then the conjunction is true. For each relational-term, the relational-primary immediately contained in it is evaluated, its truth value reversed if and only if NOT is also immediately contained in the term, and the resulting value assigned to the relational-term. A relational-primary shall be evaluated according to the description above of the various relations, if it is a comparison. Otherwise, it shall take on the value of the relational-expression immediately contained within it. This relational-expression shall be evaluated by re-applying the rules of this paragraph to it.

8.1.5 Exceptions

None.

8.1.6 Remarks

The specification for evaluation of relational-expressions guarantees that certain parts of the expression will not be evaluated if not necessary. For instance, if an array A has subscripts from 1 to 10:

```
1 <= X AND X <= 10 AND A(X) = KEY
```

will never cause an exception for subscript out of range.

8.2 Control Statements

8.2.1 General Description

Control statements allow for the interruption of the normal sequence of execution of statements by causing execution to continue at a specified line, rather than at the one with the next higher line-number.

The goto-statement allows for an unconditional transfer. The on-goto-statement allows control to be transferred to a selected line. The gosub-statement and return-statement allow for subroutine calls. The on-gosub-statement and return-statement allow for selected subroutine calls.

8.2.2 Syntax

```
1. control-transfer     = gosub-statement / goto-statement /
                          if-statement / io-recovery /
                          on-gosub-statement /
                          on-goto-statement
2. goto-statement       = (GOTO / GO TO) line-number
3. on-goto-statement    = ON index (GOTO / GO TO)
                          line-number (comma line-number)*
                          (ELSE imperative-statement)?
4. gosub-statement      = (GOSUB / GO SUB) line-number
5. return-statement     = RETURN
6. on-gosub-statement   = ON index (GOSUB / GO SUB)
                          line-number (comma line-number)*
                          (ELSE imperative-statement)?
```

8.2.3 Examples

```
2. GO TO 999
   GOTO 999
3. ON L+1 GO TO 400, 400, 500
   ON X GO TO 100, 200, 150, 9999 ELSE LET A = 1
4. GO SUB 5000
   GOSUB 5160
6. ON A+7 GOSUB 1000, 2000, 7000, 4000
   ON F1-2 GOSUB 4360, 4460, 4660 ELSE PRINT F$
```

8.2.4 Semantics

Execution of a goto-statement shall cause execution of the program to be continued at the line with the specified line-number.

The index in an on-goto-statement shall be evaluated and its value rounded to obtain an integer, whose value shall be used to select a line-number from the list following the GOTO (the line-numbers in the list are indexed from left to right, starting with 1). Execution of the program shall continue at the line with the selected line-number. If the on-goto-statement contains an ELSE clause, and the value of the index in the on-goto-statement is less than one or greater than the number of line-numbers in the list, then the imperative-statement following the ELSE shall be executed; if the imperative-statement in the ELSE part does not transfer control to another line, then execution shall be continued in sequence (i.e., with the line following that containing the on-goto-statement).

The execution of the gosub-statement or on-gosub-statement and the return-statement can be described in terms of stacks of line-numbers, one associated with each invocation of a program-unit or internal-proc-def (but may be implemented in some other fashion). (The stack is conceptual; the standard does not require that this method be used.) Prior to execution of the first gosub-statement or on-gosub-statement in the invocation of a program-unit or internal-proc-def, the stack in that entity shall be empty. Each time a gosub-statement is executed, the line-number of the gosub-statement shall be placed on top of this stack and execution of the program-unit or internal-proc-def shall be continued at the line specified in the gosub-statement.

The index in an on-gosub-statement shall be evaluated by rounding to obtain an integer, whose value shall be used to select a line-number from the list following the GOSUB (the numbers in the list are indexed from left to right, starting with 1). The line-number of the on-gosub statement shall be placed on top of the stack for the appropriate program-unit or internal-proc-def, and execution shall continue at the line with the line-number selected by the index. If the on-gosub-statement contains an ELSE clause, and the value of the index in the on-gosub-statement is less than one or greater than the number of line-numbers in the list, then the imperative-statement following the ELSE shall be executed and the stack of line-numbers shall not be changed; if the imperative-statement in the ELSE part does not transfer control to another line, execution shall then continue in sequence (i.e., with the line following that containing the on-gosub-statement).

Each time a return-statement is executed, the line-number on top of the stack shall be removed from the stack and execution of the program-unit or internal-proc-def shall continue at the line following the one with that line-number.

A return-, gosub-, and on-gosub-statement within an internal-proc-def shall interact only with the stack for that internal-proc-def. All other such statements interact only with the stack for the program-unit containing the statement.

It is not necessary that equal numbers of gosub-statements or on-gosub-statements and return-statements be executed before termination of a program-unit or internal-proc-def; the stack of line-numbers associated with the current invocation of a program-unit or internal-proc-def shall be emptied upon termination of that program-unit or internal-proc-def.

8.2.5 Exceptions

The value of the index in an on-goto-statement or an on-gosub-statement without an ELSE clause is less than one or greater than the number of line-numbers in the list (10001, fatal).

An attempt is made to execute a return-statement without having executed a corresponding gosub-statement or on-gosub-statement within the same program-unit or internal-proc-def (10002, fatal).

8.2.6 Remarks

The syntactic element control-transfer is defined solely to permit describing limitations on transfers to line numbers. It is not generated by other productions.

References to nonexistent line-numbers in a program-unit, including those in control-transfers, are syntax errors (see 4.2). There is, therefore, no exception defined in this standard for such references. Implementations may, however, choose to treat them as exceptions, if they are so documented, since the effect of nonstandard programs is implementation-defined.

8.3 Loop Structures

8.3.1 General Description

Loops provide for the repeated execution of a sequence of statements. Do-loops provide for the construction of loops with arbitrary exit conditions. The for-statement and next-statement provide for the construction of counter-controlled loops.

8.3.2 Syntax

```
 1. loop                = do-loop / for-loop
 2. do-loop             = do-line do-body
 3. do-line             = line-number do-statement tail
 4. do-statement        = DO exit-condition?
 5. exit-condition      = (WHILE / UNTIL) relational-expression
 6. do-body             = block* loop-line
 7. exit-do-statement   = EXIT DO
 8. loop-line           = line-number loop-statement tail
 9. loop-statement      = LOOP exit-condition?
10. for-loop            = for-line for-body
11. for-line            = line-number for-statement tail
12. for-statement       = FOR control-variable equals-sign
                          initial-value TO limit
                          (STEP increment)?
13. control-variable    = simple-numeric-variable
14. initial-value       = numeric-expression
15. limit               = numeric-expression
16. increment           = numeric-expression
17. for-body            = block* next-line
18. exit-for-statement  = EXIT FOR
19. next-line           = line-number next-statement tail
20. next-statement      = NEXT control-variable
```

The control-variable in the next-statement that terminates a for-loop shall be the same as the control-variable in the for-statement that begins the for-loop.

A for-loop contained in the for-body of another for-loop shall not employ the same control-variable as that other for-loop. No line-numbers in a control-transfer outside a for-loop or do-loop shall refer to a line in the for-body of that for-loop or in the do-body of that do-loop.

An exit-do-statement may only occur in a do-loop. An exit-for-statement may only occur in a for-loop.

8.3.3 Examples

```
2. 10 DO WHILE I <= N AND A(I) <> 0
   20    LET I = I + 1
   30 LOOP

2. 100 DO
   110       LET I = I+1
   120       PRINT "MORE ENTRIES (ENTER 'NO' IF NONE)"
   130       INPUT A$(I)
   140 LOOP UNTIL A$(I) = "NO"

2. 10 DO
   20    INPUT X
   30 IF 0 < X AND X <= 7 AND X = INT(X) THEN EXIT DO
   40    PRINT "INPUT AN INTEGER BETWEEN 1 AND 7"
   50 LOOP

10. 100 FOR I = 1 TO 10
    150     LET A(I) = I
    200 NEXT I

12. FOR I = A TO B STEP -1

20. NEXT C7
```

8.3.4 Semantics

An exit-condition shall be said to require exit from a loop if the value of the relational-expression following the keyword WHILE is false or if the value of the relational expression following the keyword UNTIL is true.

If execution of a program reaches a do-line, then the exit-condition, if any, in that do-line shall be evaluated. If there is no exit-condition, or if it does not require exit from the loop, then execution shall proceed to the next line. If the condition requires exit from the loop, then execution shall continue at the line following the associated loop-line. If execution of a program reaches a loop-line, then the exit-condition in that loop-line, if any, shall be evaluated. If there is no exit condition, or if it does not require exit from the loop, then execution shall resume at the associated do-line; if the condition requires exit from the loop, then execution shall continue at the line following the loop-line.

The action of the for-statement and the next-statement is defined in terms of other statements, as follows.

```
110 FOR v = initial-value TO limit STEP increment
        (lines)
150 NEXT v
```

shall be equivalent to

```
110 LET own1 = limit
120 LET own2 = increment
130 LET v = initial-value
140 DO UNTIL  (v-own1) * SGN(own2) > 0
        (lines)
150     LET v = v + own2
160 LOOP
```

Here v is any simple-numeric-variable, and own1 and own2 are variables associated with the particular for-loop and not accessible to the programmer. Similarly, the line numbers 120-140, and 160 are illustrative only; the for-loop does not actually generate additional line numbers. The variables own1 and own2 shall be distinct from similar variables associated with other for-loops. In the above equivalence, a control-transfer to the for-line shall be interpreted as a control-transfer to the first let-statement, and a control-transfer to the next-line shall be interpreted as a control-transfer to the last let-statement.

In the absence of a STEP clause in a for-statement, the value of the increment shall be +1.

Execution of an exit-do-statement shall cause execution to continue at the line following the loop-line of the smallest do-loop in which the exit-do-statement occurs. Execution of the exit-for-statement shall cause execution to continue at the line following the next-line of the smallest for-loop in which the exit-for-statement occurs.

8.3.5 Exceptions

None.

8.3.6 Remarks

On exit from a for-loop through the next-statement, the value of the control-variable is the first value not used; on all other exits from a for-loop the control-variable retains its current value.

8.4 Decision Structures

8.4.1 General Description

An if-statement allows for conditional transfers, for the conditional execution of a single imperative-statement, or for the execution of one of two alternative imperative-statements.

An if-block allows for the conditional execution of a sequence of lines or for the execution of one of several alternative sequences of lines.

A select-block allows for the conditional execution of any one of a number of alternative sequences of lines, based on the value of an expression.

8.4.2 Syntax

```
 1. if-statement      = IF relational-expression
                        THEN if-clause (ELSE if-clause)?
 2. if-clause         = imperative-statement / line-number
 3. if-block          = if-then-line then-block elseif-block*
                        else-block? end-if-line
 4. if-then-line      = line-number IF relational-expression
                        THEN tail
 5. then-block        = block*
 6. elseif-block      = elseif-then-line block*
 7. elseif-then-line  = line-number ELSEIF
                        relational-expression THEN tail
 8. else-block        = else-line block*
 9. else-line         = line-number ELSE tail
10. end-if-line       = line-number END IF tail
11. select-block      = select-line remark-line* case-block
                        case-block* case-else-block?
                        end-select-line
12. select-line       = line-number select-statement tail
13. select-statement  = SELECT CASE expression
14. case-block        = case-line block*
15. case-line         = line-number case-statement tail
16. case-statement    = CASE case-list
17. case-list         = case-item (comma case-item)*
18. case-item         = constant / range
19. range             = (constant TO / IS relation) constant
20. case-else-block   = case-else-line block*
21. case-else-line    = line-number CASE ELSE tail
22. end-select-line   = line-number END SELECT tail
```

The constants appearing in case-statements in a select-block shall be the same type (i.e., either numeric or string) as the expression in the select-statement. The ranges and constants specified in case-lists in a select-block shall not overlap.

No line-number in a control-transfer outside an if-block, then-block, elseif-block, else-block, select-block, case-block, or case-else-block shall refer to a line inside that if-block, then-block, elseif-block, else-block, select-block, case-block, or case-else-block, respectively, other than to the if-then-line of that if-block or the select-line of that select-block.

A line-number in a control-transfer inside an elseif-block, else-block, case-block, or case-else-block shall not refer to the associated elseif-then-line, else-line, case-line, or case-else-line.

8.4.3 Examples

1.
```
IF X => Y2 THEN GOSUB 900 ELSE GOSUB 2000
IF X$ = "NO" OR X$ = "STOP" THEN LET A = 1
IF A = B THEN 100
IF A$ = B$ THEN 200 ELSE 300
```

3.
```
10 IF X = INT(X) THEN
20       PRINT X; "IS AN INTEGER"
30 ELSE
40       PRINT X; "IS NOT AN INTEGER"
50 END IF

100 IF A = 0 THEN
110     PRINT "ONE ROOT"
120 ELSEIF DISC < 0 THEN
130     PRINT "COMPLEX ROOTS"
140 ELSE
150     PRINT "REAL ROOTS"
160 END IF
```

11.
```
10 SELECT CASE A$(1:1)
20 CASE "A" TO "Z", "a" TO "z"
30       PRINT A$; "starts with a letter"
40 CASE "0" TO "9"
50       PRINT A$; "starts with a digit"
60 CASE ELSE
70       PRINT A$; "doesn't start with a letter or a digit"
80 END SELECT
```

```
10 SELECT CASE X
20 CASE IS < 0
30      PRINT X; "is negative"
40 CASE IS > 0
50      PRINT X; "is positive"
60 CASE ELSE
70      PRINT X; "is zero"
80 END SELECT
```

8.4.4 Semantics

If the value of the relational-expression in an if-statement is true and an imperative-statement follows the keyword THEN, then this imperative-statement shall be executed; if a line-number follows the keyword THEN, then execution of the program shall be continued at the line with that line-number. If the value of the relational-expression is false and an imperative-statement follows the keyword ELSE, then this imperative-statement shall be executed; if a line-number follows the keyword ELSE, then execution of the program shall be continued at the line with that line-number; if no ELSE is present, then execution shall be continued in sequence, i.e., with the line following that containing the if-statement.

If-blocks shall be executed as follows. If a then-block, elseif-block, or else-block does not contain a block, the effect is as if it did contain a block consisting of a remark-line. If the value of the relational-expression in the if-then-line is true, then execution shall continue at the first line of the corresponding then-block. If false, then the relational-expressions of each corresponding elseif-then-line, if any, shall be evaluated in order. As soon as a true relational-expression is found, execution shall continue at the first line of the blocks of that elseif-block. If no true relational-expression is found in the elseif-then-lines, then, if an else-block is present, execution shall continue at the first line of the block of that else-block. If there is no else-block, execution shall continue at the line following the end-if-line. When execution reaches the end of a then-block, an elseif-block, or an else-block, it shall continue at the line following the corresponding end-if-line.

The expression in a select-statement in a select-block shall be evaluated and its value compared with the case-items in the case statements until a match is found. A match shall occur when (1) the value of the expression equals that of a constant appearing as a case-item, (2) the value is greater than or equal to that of the first constant appearing in a range containing the

word TO, but less than or equal to the second, or (3) the value satisfies the relationship indicated by the relation appearing before the constant in a range. If and when a match is found, the rest of the case-block headed by the case-statement in which the match was found shall be executed. If no case-item is matched, then the case-else-block, if it is present, shall be executed. When execution reaches the end of a case-block or case-else-block, it shall continue at the line following the end-select-line.

Nesting of blocks is permitted subject to the same nesting constraints as for-loops (i.e., no overlapping blocks).

8.4.5 Exceptions

A select-block without a case-else-block is executed and no case-block is selected (10004, fatal).

8.4.6 Remarks

None.

9. Program Segmentation

BASIC provides four mechanisms for the segmentation of programs. The first provides for user-defined functions, whose values may be used in numeric- and string-expressions. The second enables subprograms to be defined, which communicate via parameters and which can be invoked via a call-statement. The third enables separate programs to be executed sequentially without user intervention. The fourth, described in 13.5, enables the definition of graphical pictures.

Functions and subprograms (which we refer to collectively as "routines") are of two types: internal and external. This distinction also applies to picture definitions as specified in 13.5. External routines are independent program-units lexically following the main-program. Internal routines are contained within a program-unit (the main-program, an external routine, or a parallel-section) and are considered to be part of that program-unit. An internal routine cannot contain another internal routine.

In general, an external routine does not share anything (including, but not limited to, variables, DATA statements, internal routines, OPTIONs, and DEBUG status) with other program-units. Information is exchanged between external routines and other program-units by means of parameters and, in the case of external functions, returned values. In general, an internal routine shares everything with its surrounding program-unit, with the exception of its parameters. There are no local variables for internal routines. See Appendix B for more detail on scope rules.

Within a program-unit, a routine shall always be defined or declared in a line lexically preceding its first invocation in that program-unit. It is not an error for a routine to be defined or declared without being invoked. An external routine may be invoked throughout the program; an internal routine may be invoked only from within its containing program-unit.

No control-transfer within an internal or external routine may refer to a line-number outside that routine, nor may a control-transfer outside a routine refer to a line-number within it.

9.1 User-Defined Functions

9.1.1 General Description

In addition to the implementation-supplied functions provided for the convenience of the programmer (cf. 5.4, 6.4 and elsewhere), BASIC allows the programmer to define new functions within a program-unit or program.

9.1.2 Syntax

```
 1. function-def              = internal-function-def /
                                external-function-def
 2. internal-function-def     = internal-def-line /
                                internal-function-line
                                block* end-function-line
 3. internal-def-line         = line-number def-statement tail
 4. def-statement             = numeric-def-statement /
                                string-def-statement
 5. numeric-def-statement     > DEF numeric-defined-function
                                function-parm-list? equals-sign
                                numeric-expression
 6. numeric-defined-function = numeric-identifier
 7. string-def-statement      = DEF string-defined-function
                                length-max? function-parm-list?
                                equals-sign string-expression
 8. string-defined-function   = string-identifier
 9. function-parm-list        = left-parenthesis
                                function-parameter
                                (comma function-parameter)*
                                right-parenthesis
10. function-parameter        > simple-variable / formal-array
11. formal-array              = array-name left-parenthesis
                                comma* right-parenthesis
12. internal-function-line    > line-number FUNCTION
                                (numeric-defined-function /
                                (string-defined-function
                                length-max?))
                                function-parm-list? tail
13. end-function-line         = line-number END FUNCTION tail
14. external-function-def     = external-function-line
                                unit-block* end-function-line
15. external-function-line    > line-number EXTERNAL FUNCTION
                                (numeric-defined-function /
                                (string-defined-function
                                length-max?))
                                function-parm-list? tail
```

```
16. numeric-function-let-statement = LET numeric-defined-function
                                     equals-sign numeric-expression
17. string-function-let-statement  = LET string-defined-function
                                     equals-sign string-expression
18. exit-function-statement        = EXIT FUNCTION
19. type-declaration               > def-type /
                                     internal-function-type /
                                     external-function-type
20. def-type                       = DEF function-list
21. internal-function-type         = FUNCTION function-list
22. external-function-type         = EXTERNAL FUNCTION function-list
23. function-list                  = defined-function
                                     (comma defined-function)*
24. defined-function               > numeric-defined-function /
                                     string-defined-function
```

No line-number in a control-transfer outside an internal-function-def shall refer to a line in an internal-function-def other than to an internal-function-line, nor shall a line-number in a control-transfer inside an internal-function-def refer either to a line outside that internal-function-def or to the associated internal-function-line.

A line-number in a control-transfer inside an external-function-def shall not refer to the associated external-function-line.

If a defined-function is defined by an external-function-def, it shall not be defined more than once in the program. If a defined-function is defined by an internal-function-def, it shall not be defined more than once in the containing program-unit.

Within a program-unit, no more than one function (internal or external) of a given name shall be declared or defined.

If a defined-function is defined by an external-function-def, then a declare-statement with external-function-type containing that defined-function shall occur in a lower-numbered line than the first reference to that defined-function in the same program-unit.

If a defined-function is defined by an internal-function-def other than an internal-def-line, then either the internal-function-def, or a declare-statement with internal-function-type naming that defined-function, shall occur in a lower-numbered line than the first reference to that defined-function in the same program-unit.

If a defined-function is defined by an internal-def-line, then either the internal-def-line, or a declare-statement with def-type naming that defined-function, shall occur in a lower-numbered line than the first reference to that defined-function in the same program-unit.

Self-recursive functions need not declare themselves; that is, if a function-def contains a reference to itself, that reference does not require a type-declaration containing the defined-function in a lower-numbered line.

An exit-function statement shall occur only within a function-def.

Within each function-def (other than an internal-def-line) shall occur at least one numeric- or string-function-let-statement with defined-function the same as the defined-function in the internal- or external-function-line of the function-def.

The number and type of function-arguments in a numeric-function-ref or string-function-ref shall agree with the number and type of function-parameters in the corresponding function-def. That is,

(1) The number of function-arguments shall be the same as the number of function-parameters.

(2) The function-arguments in the function-arg-list shall be associated with the corresponding function-parameters in the function-parm-list (i.e., the first with the first, the second with the second, and so on), and the types shall correspond as follows:

Parameter	Argument
simple-numeric-variable	numeric-expression
simple-string-variable	string-expression
formal-array (numeric)	actual-array (numeric)
formal-array (string)	actual-array (string)

The number of dimensions of an actual-array shall be one more than the number of commas in the corresponding formal-array. A formal-array shall have no more than three dimensions (two commas).

Whenever a numeric argument is passed to a corresponding numeric parameter in a different program-unit, the ARITHMETIC options in effect for the two program-units shall agree.

The ARITHMETIC option of a external-function-def of numeric type shall agree with that of the invoking program-unit.

A given function-parameter shall occur only once in a function-parm-list. Function-parameters shall not be explicitly declared or dimensioned within the internal- or external-function-def.

A defined-function appearing in a def-type or internal-function-type shall be defined elsewhere in the same program-unit by an internal-def-line or internal-function-def (other than an internal-def-line), respectively.

A defined-function appearing in an external-function-type shall be defined elsewhere in the program by an external-function-def.

9.1.3 Examples

```
 5. DEF E = 2.7182818
    DEF AVERAGE(X,Y) = (X+Y)/2
 7. DEF FNA$(S$,T$) = S$ & T$
    DEF Right$(A$, n) = A$( Len(A$)-n+1 : Len(A$) )
14. 100 EXTERNAL FUNCTION ANSWER(A$)
    120     SELECT CASE UCASE$(A$)
    130     CASE "YES"
    140          LET ANSWER=1
    150     CASE "NO"
    160          LET ANSWER=2
    170     CASE ELSE
    180          LET ANSWER=3
    190     END SELECT
    200 END FUNCTION
21. FUNCTION AVERAGE, REVERSE$
```

9.1.4 Semantics

A function-def specifies the means of evaluating a function based on the values of the parameters appearing in the function-parm-list and possibly other variables or constants.

9.1.4.1 Function Parameters. When a defined-function is referenced (i.e., when an expression involving the function is evaluated), then the arguments in the function reference, if any, shall be evaluated from left to right and their values shall be assigned to the parameters in the function-parm-list for the function-def (i.e., arguments shall be passed by value to the parameters of the function). The number of dimensions in a

formal-array is one more than the number of commas in the formal-array. Upon invocation of a function-def, a formal-array has the same bounds as the corresponding actual-array. A simple-string-variable or string-array that is a function-parameter shall have the implementation-defined default as its maximum length.

9.1.4.2 Function Evaluation. If a function is defined in a def-statement, then the expression in that statement shall be evaluated and its value assigned as the value of the function. If a function is defined in an internal- or external-function-def, then the lines following the internal- or external-function-line shall be executed in sequential order until one of the following occurs:

(1) Some other action is dictated by execution of a line
(2) A fatal exception occurs
(3) A chain- or stop-statement is executed
(4) An exit-function-statement is executed
(5) An end-function-line is reached

The value of the defined-function shall be set by execution of one or more numeric-function-let-statements or string-function-let-statements. Upon exit from the function-def, the value shall be that most recently assigned to the defined-function in that invocation. If, upon exit, no such value has been assigned, then the result shall be consistent with the implementation-defined policies for uninitialized variables. A length-max following a string-defined-function establishes the maximum length of the string value to be returned by that function-def. If no length-max is specified, then the maximum length shall be the same as for a string-variable without a length-max.

An exit-function-statement, when executed, shall terminate the execution of the function-def in which it is immediately contained. An end-function-line marks the textual end of a function-block, and also shall terminate execution of the function-block. Execution of a stop-statement in a function-block shall terminate execution of the entire program.

A function-def may refer, directly or indirectly, to the function being defined (i.e., recursive function invocations are permitted).

Lines in a function-def shall not be executed unless the function it defines is referenced. If the execution of a program reaches an internal-def-line, it shall proceed to the next line

without further effect. If execution reaches an internal-function-line, it shall proceed to the line following the associated end-function-line without further effect.

9.1.4.3 Scopes of Variables, Arrays, Channel-Numbers, and Data. A function-parameter appearing in the function-parm-list of a function-def shall be local to each invocation of that function-def (i.e., it shall name a variable or array distinct from any variable or array with the same name outside the function-def).

The treatment of variables and arrays that are not named as function-parameters in a function-def shall depend upon whether the function-def is internal or external. If the function-def is external, then such variables and arrays shall be local to each invocation of that program-unit (i.e., they shall be distinct from objects with the same names outside that function-def or within other invocations of that function-def); in addition, they shall be initialized or not initialized in a manner consistent with the implementation-defined policies for the main-program or parallel-section each time the function-def is invoked. If the function-def is internal, then those variables and arrays shall be global to the containing program-unit and shall retain their assigned values each time the function-def is invoked; if these values are changed during the course of executing the internal-function-def, the changes remain in effect when execution is returned to the surrounding program-unit.

With one exception, the scope of channel-numbers (cf. 9.2) is always the program-unit. Nonzero channel-numbers within a function-def shall be local to each invocation of that function-def if it is external, and shall be global to the containing program-unit in which it occurs if it is internal. Channel zero shall be global to the entire program. Files shall be assigned to nonzero channels within a program-unit by means of an open-statement before use. Files assigned to channels local to a function-def shall be closed upon exit from that function-def.

The scope of internal data is always the program-unit. Thus, data within an external-function-def shall be local to each invocation of that program-unit. Hence read-statements and restore-statements within such a function-def shall refer only to data in data-statements within that function-def and not to data in other program-units. Upon invocation of such a function-def, the pointer for the data within that function-def shall be reset to the beginning of the data (cf. 10.1). Data within an internal-function-def shall be part of the data sequence for the containing program-unit, and read-statements and restore-

statements within such a function-def shall refer to the entire sequence of data in that program-unit.

9.1.5 Exceptions

A string-function-let-statement attempts to assign a value whose length exceeds the maximum for the string-defined-function (1106, fatal).

9.1.6 Remarks

Incompatible COLLATE options are allowed between an invoking and invoked program-unit (even if they communicate via string parameters) because COLLATE does not dictate the internal representation of strings, but only their order in string comparisons and the values of the CHR$ and ORD functions.

It is not an error for a function to be defined by an internal-function-def or to appear in a declare-statement, but not to be referred to in that program-unit.

It is not an error for an internal-function-def to appear before a declare-statement with def-type or internal-function-type containing the name of that internal function.

An internal-function-type or def-type may be omitted if the corresponding definition appears before the first reference to that function. An external-function-type is always required when an external-function is referenced in a program-unit other than its own.

The requirement that both internal and external functions be declared or defined before they are used allows several program-units within a program each to contain an internal function with the same name as an external function. This facilitates the use of function libraries in which the programmer may not know the names of all the external-function-defs in the library.

9.2 Subprograms

9.2.1 General Description

Subprograms provide a mechanism for the logical segmentation of programs, allowing parameters to be passed between program segments. Subprograms, like defined-functions, may be internal or external to a program-unit.

9.2.2 Syntax

```
 1. subprogram-def           = internal-sub-def /
                               external-sub-def
 2. internal-sub-def         = internal-sub-line block*
                               end-sub-line
 3. internal-sub-line        = line-number sub-statement tail
 4. sub-statement            = SUB subprogram-name
                               procedure-parm-list?
 5. subprogram-name          = routine-identifier
 6. procedure-parm-list      = left-parenthesis
                               procedure-parameter
                               (comma procedure-parameter)*
                               right-parenthesis
 7. procedure-parameter      > simple-variable / formal-array /
                               channel-number
 8. channel-number           = number-sign integer
 9. end-sub-line             = line-number end-sub-statement tail
10. end-sub-statement        = END SUB
11. exit-sub-statement       = EXIT SUB
12. external-sub-def         = external-sub-line unit-block*
                               end-sub-line
13. external-sub-line        = line-number EXTERNAL
                               sub-statement tail
14. call-statement           = CALL subprogram-name
                               procedure-argument-list?
15. procedure-argument-list  = left-parenthesis
                               procedure-argument
                               (comma procedure-argument)*
                               right-parenthesis
16. procedure-argument       = expression / actual-array /
                               channel-expression
17. type-declaration         > internal-sub-type /
                               external-sub-type
18. internal-sub-type        = SUB sub-list
19. external-sub-type        = EXTERNAL SUB sub-list
20. sub-list                 = subprogram-name (comma
                               subprogram-name)*
```

No line-number in a control-transfer outside an internal-sub-def shall refer to a line in an internal-sub-def other than to an internal-sub-line, nor shall a line-number in a control-transfer inside an internal-sub-def refer either to a line outside that internal-sub-def or to the associated internal-sub-line.

A line-number in a control-transfer inside an external-sub-def shall not refer to the associated external-sub-line.

If a subprogram-name is defined by an external-sub-def, it shall not be defined more than once in the program. If a subprogram-name is defined by an internal-sub-def, it shall not be defined more than once in the containing program-unit.

Within a program-unit, no more than one subprogram (internal or external) of a given name shall be declared or defined.

If a subprogram-name is defined by an external-sub-def, then a declare-statement with external-sub-type containing that subprogram-name shall occur in a lower-numbered line than the first reference to that subprogram-name in a call-statement in the same program-unit.

If a subprogram-name is defined by an internal-sub-def, then either the internal-sub-def, or a declare-statement with internal-sub-type containing that subprogram-name, shall occur in a lower-numbered line than the first reference to that subprogram-name in the same program-unit.

Self-recursive subprograms need not declare themselves; that is, if a subprogram-def contains a reference to itself in a call-statement, that reference does not require a type-declaration containing that subprogram-name in a lower-numbered line.

An exit-sub-statement shall occur only within a subprogram-def.

The number and type of procedure-arguments in a call-statement shall agree with the number and type of procedure-parameters in the corresponding subprogram-def. That is,

(1) The number of procedure-arguments shall be the same as the number of procedure-parameters.

(2) The procedure-arguments in the procedure-argument-list shall be associated with the corresponding procedure-parameters in the procedure-parm-list (i.e., the first with the first, the

second with the second, and so on), and the types shall correspond as follows:

Parameter	Argument
simple-numeric-variable	numeric-expression
simple-string-variable	string-expression
formal-array (numeric)	actual-array (numeric)
formal-array (string)	actual-array (string)
channel-number	channel-expression

An actual-array shall have the same number of dimensions as the corresponding formal-array. The number of dimensions in a formal-array is one more than the number of commas in the formal-array.

Whenever a numeric argument is passed to a corresponding numeric parameter in a different program-unit, the ARITHMETIC options in effect for the two program-units shall agree.

A given procedure-parameter shall occur only once in a procedure-parm-list. Procedure-parameters shall not be explicitly declared or dimensioned within the internal- or external-sub-def.

The channel-number #0 shall not be used as a procedure-parameter.

A subprogram-name appearing in an internal-sub-type shall be defined elsewhere in the same program-unit by an internal-sub-def.

A subprogram-name appearing in an external-sub-type shall be defined elsewhere in the program by an external-sub-def.

9.2.3 Examples

```
 2. 100 SUB exchange(a,b)
    110    LET t = a
    120    LET a = b
    130    LET b = t
    140 END SUB
 4. SUB CALC(X,Y,Z$)
    SUB SORT(A(),B(,),A$,#3)
13. 2000 EXTERNAL SUB OPEN (#1, fname$, result)
14. CALL CALC (3*A+2, 7715, "NO")
    CALL SORT (Zvect, Ymat, (L$), #N)
```

9.2.4 Semantics

When a call-statement is executed, control shall be transferred to the subprogram named in the call-statement. Execution of the subprogram shall begin at the line following the sub-line and shall continue in sequential order until one of the following occurs:

(1) Some other action is dictated by execution of a line
(2) A fatal exception occurs
(3) A chain-statement is executed
(4) A stop- or exit-sub-statement is executed
(5) An end-sub-line is reached

The end-sub-line serves both to mark the textual end of a subprogram and, when executed, to terminate execution of the subprogram. The exit-sub-statement, when executed, shall terminate the execution of the innermost subprogram in which it is contained. When execution of a subprogram terminates, execution shall continue at the line following the call-statement that initiated execution of the subprogram.

Execution of a stop-statement in a subprogram shall terminate execution of the entire program.

A subprogram may call itself, either directly or indirectly through another procedure (i.e., recursive subprogram invocations are permitted).

Lines in a subprogram-def shall not be executed unless the subprogram it defines is referenced through a call-statement. If execution reaches an internal-sub-line, it shall proceed to the line following the associated end-sub-line without further effect.

9.2.4.1 Subprogram Parameters. When a call-statement is executed, its procedure-arguments shall be identified, from left to right, with the corresponding procedure-parameters in the sub-statement for the subprogram.

Procedure-arguments that are numeric-variables or string-variables without substring-qualifiers shall be passed by reference (i.e., any reference to the corresponding procedure-parameter within the subprogram shall result in a reference to the procedure-argument, and any assignment to the procedure-parameter shall result in an assignment to the corresponding procedure-argument).

If a procedure-argument is an array element, its subscripts shall be evaluated once at each entry to the subprogram.

A procedure-argument that is an expression, but not a numeric-variable or a string-variable without a substring-qualifier, shall be evaluated once at each entry to the subprogram and the value so obtained shall be assigned to a location local to the subprogram. This local value shall be used in any reference to the corresponding procedure-parameter, and this local location shall be used as the destination of any assignment to the procedure-parameter. Any necessary evaluation of procedure-arguments shall take place from left to right.

References within a subprogram to procedure-parameters that are formal-arrays shall result in references to the corresponding arrays in the procedure-argument-list; assignments to or redimensioning of such arrays shall result in assignments to or redimensioning of the corresponding arrays in the procedure-argument-list. Upon entry to the subprogram, a formal-array as a procedure-parameter has the same bounds as the corresponding procedure-argument.

For a procedure-parameter that is a simple-string-variable or string-array, the associated maximum length shall be the implementation-defined default, in the case of passing by value; when passing by reference, the maximum length shall be that of the corresponding procedure-argument.

If both an array and one of its elements are named as procedure-arguments in a call-statement and the array is redimensioned during execution of the subprogram, then any subsequent reference within the subprogram to the procedure-parameter associated with the array-element shall produce implementation-defined results.

A procedure-argument that is a channel-expression shall be evaluated once on entry to the subprogram and the resulting channel shall be used whenever the value of the corresponding procedure-parameter is referenced in the subprogram. The attributes of the file (cf. 11.1.4) assigned to this channel shall be passed unchanged to the subprogram, and changes to attributes and contents of the file within the subprogram shall be immediately effective, regardless of which of the channel-numbers is used in the subsequent reference, and shall remain in effect upon exit from the subprogram.

A file need not be assigned to a channel designated by a procedure-argument when a call-statement is executed. If an

open-statement within a subprogram assigns a file to that channel, then that assignment shall remain in effect upon exit from the subprogram.

9.2.4.2 Scopes of Variables, Arrays, Channel-Numbers, and Data. A procedure-parameter appearing in the procedure-parm-list of a subprogram-def that has been passed by value shall be local to each invocation of the subprogram-def (i.e., it shall name a variable or array distinct from any variable or array with the same name outside the subprogram-def).

For a procedure-parameter that has been passed by reference, its name shall be local to each invocation of the subprogram-def, but that name refers to the same object as the corresponding procedure-argument (see 9.2.4.1).

The treatment of variables and arrays that are not named as parameters in a subprogram-def shall depend upon whether the subprogram-def is internal or external. If the subprogram-def is external, then such variables and arrays shall be local to each invocation of that program-unit; that is, they shall be distinct from objects with the same names outside that subprogram-def or within other invocations of that subprogram-def; in addition, they shall be initialized or not initialized in a manner consistent with the implementation-defined policies for the main-program or parallel-section each time the subprogram-def is invoked. If the subprogram-def is internal, then those variables and arrays shall be global to the containing program-unit and shall retain their assigned values each time the subprogram-def is invoked; if these values are changed during the course of executing the subprogram-def, the changes remain in effect when execution is returned to the surrounding program-unit.

With one exception, the scope of channel-numbers that are not procedure-parameters is always the program-unit. Nonzero channel-numbers within a subprogram-def shall be local to each invocation of that subprogram-def if it is external, and shall be global to the program-unit in which it occurs if it is internal. Channel zero shall be global to the entire program. Files shall be assigned to nonzero channels within a program-unit by means of an open-statement before use. Files assigned to channels local to a subprogram-def shall be closed upon exit from that subprogram-def.

The scope of internal data is always the program-unit. Thus, data within an external-sub-def shall be local to each invocation of that program-unit. Hence read-statements and restore-statements within such a subprogram-def shall refer only

to data in data-statements within that subprogram-def and not to data in other program-units. Upon invocation of such a subprogram-def, the pointer for the data within that subprogram-def shall be reset to the beginning of the data (cf. 10.1). Data within an internal-sub-def shall be part of the data sequence for that program-unit, and read-statements and restore-statements within such a subprogram-def shall refer to the entire sequence of data in that program-unit.

9.2.5 Exceptions

None.

9.2.6 Remarks

Implementations may extend the language by making the use of an internal-sub-type optional, even when the internal-sub-defs occur after the call-statements referring to them.

An alias is said to exist for an object whenever two or more distinct names exist for that object within the same scope. When parameters are passed by reference, aliases may be created in certain circumstances. Parameter passing by value does not create aliases, since distinct objects are created for each parameter.

Any call-statement creates aliases whenever:

(1) Channel-expressions that round to the same integer value are passed to different formal channel-numbers

(2) The same actual-array is passed to different formal-arrays

(3) The same simple variable or array element is passed to different formal simple-variables

(4) An array is passed to a formal-array and an element of that array is passed to a formal simple-variable

(5) A channel-expression is passed to an internal subprogram

(6) An argument that is not a channel-expression is passed by reference to an internal subprogram

In the first four cases, the alias arises because two or more formal parameters name the same object, or parts of the same

object. In the latter two cases, the alias arises because an object is "visible" to an internal subprogram, both as a parameter and as an object global to the entire program-unit.

When the state of an object referred to by an aliased procedure-parameter is changed, that change shall be immediately effective in every subsequent reference to the object, regardless of which of the object's names is used in the reference. Events that potentially affect the state of the object referred to by a procedure-parameter include assignment, input/output operations, and array redimensioning.

Thus, the program:

```
100  DECLARE INTERNAL SUB S
110  LET A = 0
120  CALL S(A,A)
130  SUB S(B,C)
140      LET A = 1
150      LET B = 2
160      LET C = 3
170      IF A <> B OR B <> C OR A <> C THEN
180         PRINT "This shouldn't happen."
190      END IF
200  END SUB
210  END
```

would never print the error message in a conforming implementation.

Remarks about the following topics in 9.1.6 apply analogously to subprograms:

(1) Program-units with different COLLATE options

(2) Functions that are defined or declared, but not referenced

(3) Functions that are defined before they are declared

(4) The requirement that external, but not internal, functions always be declared (rather than defined)

(5) Internal functions with the same name in different program-units

9.3 Chaining

9.3.1 General Description

The chain-statement allows separate programs to be executed serially without programmer intervention. Such a facility is useful for segmenting large programs.

9.3.2 Syntax

1. chain-statement = CHAIN program-designator (WITH function-arg-list)?
2. program-designator = string-expression

The association of the function-arguments in the function-arg-list in the chain-statement with the function-parameters in the function-parm-list in the program-name-line shall follow the same rules set down for defined-functions (see 9.1).

9.3.3 Examples

1.
```
CHAIN "PROG2"
CHAIN A$ WITH (X,FILENAME$)
```

9.3.4 Semantics

A chain-statement shall terminate execution of the current program, close all files, and initiate execution of the program designated by the program-designator. The way in which a program is associated with its program-designator is implementation-defined.

If the program being chained to contains a program-name-line, then the arguments of the chain-statement are evaluated and assigned to the corresponding parameters in the program-name-line (i.e., parameters are passed by value). The bounds of a formal-array shall therefore be adjusted to equal those of the corresponding actual-array, in accordance with the rules for passing array parameters to functions (cf. 9.1).

It is implementation-defined whether upper-case- and lower-case-letters are treated as equivalent in a program-designator.

The initial values of variables in a chained-to program are implementation-defined.

9.3.5 Exceptions

The program identified by the program-designator is not available (10005, fatal).

The number and type of arguments in a chain-statement do not agree with the number and type of the corresponding parameters in the program-name-line of the program being chained to, or a program-name-line with a function-parm-list is not present (4301, fatal).

An actual-array does not have the same number of dimensions as the corresponding formal-array (4302, fatal).

Numeric parameters are passed between programs with a chain-statement and the ARITHMETIC options of the program-units disagree (4303, fatal).

9.3.6 Remarks

In a typical implementation, a program-designator will be the name of a file containing that program. The program chained to need not be a BASIC program.

If exception 4301, 4302, or 4303 occurs, it may be reported by the chained-from program, the chained-to program, or some intermediate system program.

10. Input and Output

Input and output facilities are provided for the interaction of a BASIC program with collections of data. Data may be obtained by a program from statements within that program, from a standard source external to that program, or from a named source external to that program (cf. 11.4). Output data may be directed to a standard destination external to that program or to a named destination external to that program (cf. 11.3 and 11.5).

10.1 Internal Data

10.1.1 General Description

The read-statement provides for the assignment of values to variables from a sequence of data created from one or more data-statements. The restore-statement allows data in a program to be reread.

10.1.2 Syntax

```
1. read-statement      > READ (missing-recovery colon)?
                         variable-list
2. variable-list       = variable (comma variable)*
3. missing-recovery    = IF MISSING THEN io-recovery-action
4. io-recovery-action  = exit-do-statement /
                         exit-for-statement / line-number
5. restore-statement   = RESTORE line-number?
6. data-statement      = DATA data-list
7. data-list           = datum (comma datum)*
8. datum               = constant / unquoted-string
9. unquoted-string     = plain-string-character /
                         plain-string-character
                         unquoted-string-character*
                         plain-string-character
```

An io-recovery-action containing an exit-for-statement shall occur only within a for-body. An io-recovery-action containing an exit-do-statement shall occur only within a do-body.

If a line-number occurs in a restore-statement, the line-number shall refer to a line containing a data-statement.

10.1.3 Examples

```
1. READ X, Y,Z
   READ IF MISSING THEN 1350: X(1), A$
```

```
5. RESTORE
   RESTORE 1000
6. DATA 3.14159, PI, 5E-30, ","
9. COMMAS CANNOT OCCUR IN UNQUOTED STRINGS.
```

10.1.4 Semantics

Data from the totality of data-statements in each program-unit shall behave as if collected into a single data sequence. The order in which data appear textually in the totality of all data-statements determines the order of the data in the data sequence.

If the execution of a program reaches a line containing a data-statement, then it shall proceed to the next line with no further effect.

Execution of a read-statement shall cause variables in the variable-list to be assigned values, in order, from the sequence of data in the program-unit containing the read-statement. A conceptual pointer is associated with this data sequence. At the initiation of execution of the program-unit, this conceptual pointer points to the first datum in the data sequence. Each time a read-statement is executed, each variable in the variable-list in sequence is assigned the value of the datum indicated by the pointer and the pointer advanced to point beyond that datum.

If an attempt is made to read data beyond the end of the data sequence, an exception shall occur unless a missing-recovery is present in the read-statement. In that case, the specified io-recovery-action shall be taken. If the io-recovery-action is an exit-do- or exit-for-statement, that statement shall have its normal effect (cf. 8.3). If the io-recovery-action is a line-number, then execution shall continue at the line having that line-number.

The type of a datum in the data sequence shall correspond to the type of the variable to which it is to be assigned(i.e., numeric-variables require numeric-constants as data and string-variables require string-constants or unquoted-strings as data). An unquoted-string that is also a numeric-constant may be assigned to either a string-variable or a numeric-variable by a read-statement.

If the evaluation of a numeric datum causes an underflow, then its value shall be replaced by zero.

Subscripts and substring-qualifiers in the variable-list shall be evaluated after values have been assigned to the variables preceding them (i.e., to the left of them) in the variable-list.

Execution of a restore-statement resets the pointer for the data sequence in the program-unit containing the restore-statement to the beginning of the sequence, so that the next read-statement executed shall read data from the beginning of the sequence. If a line-number is present, then the pointer for the data sequence in the program-unit containing the restore-statement is set to the first datum in the data-statement with the given line-number, so that the next read-statement executed shall read data from the beginning of the designated data-statement.

10.1.5 Exceptions

The variable-list in a read-statement requires more data than are present in the remainder of the data-list and a missing-recovery has not been specified (8001, fatal).

An attempt is made to assign a value to a numeric-variable from a datum that is not a numeric-constant (8101, fatal).

The evaluation of a numeric datum causes an overflow (1006, fatal).

The assignment of a datum to a string-variable results in a string overflow (1053, fatal).

10.1.6 Remarks

Implementations may choose to treat underflows as exceptions (1506, nonfatal: supply zero and continue) to permit interception by exception handlers.

10.2 Input

10.2.1 General Description

Input-statements provide for user interaction with a program by allowing variables to be assigned values supplied from a source external to the program. The input-statement enables the entry of mixed string and numeric data, with data items being separated by commas.

A prompt for input may be specified to replace the usual prompt supplied by the implementation.

The line-input-statement enables an entire line of input, including embedded spaces and commas, to be assigned as the value of a string-variable.

10.2.2 Syntax

```
 1. input-statement        > INPUT input-modifier-list?
                             variable-list
 2. input-modifier-list    = input-modifier
                             (comma input-modifier)* colon
 3. input-modifier         = prompt-specifier /
                             timeout-expression /
                             time-inquiry
 4. prompt-specifier       = PROMPT string-expression
 5. timeout-expression     = TIMEOUT numeric-time-expression
 6. numeric-time-expression= numeric-expression
 7. time-inquiry           = ELAPSED numeric-variable
 8. line-input-statement   > LINE INPUT input-modifier-list?
                             string-variable-list
 9. input-prompt           = [implementation-defined]
10. input-reply            = data-list comma? end-of-line
11. line-input-reply       = character* end-of-line
```

At most one prompt-specifier, one timeout-expression, and one time-inquiry shall occur in an input-modifier-list. These may occur in any sequence.

10.2.3 Examples

```
 1. INPUT X
    INPUT X, A$, Y(2)
    INPUT PROMPT "What is your name? ": N$
    INPUT TIMEOUT 3*N, ELAPSED T, PROMPT Pstring$: N$
 8. LINE INPUT A$
    LINE INPUT PROMPT "": A$, B$
```

```
10. 2, SMITH, -3
    25, 0, -10.2
11. He said, "Don't".
```

10.2.4 Semantics

Execution of an input-statement shall cause execution of the program to be suspended until a valid input-reply, as specified below, has been supplied. An input-statement shall cause variables in the variable-list to be assigned, in order, values from the input-reply.

In interactive mode, the user of the program shall be informed of the need to supply data by the output of an input-prompt.

10.2.4.1 Input-Modifier-List. If a prompt-specifier is present in the input-statement, then the implementation-defined input-prompt shall not be output; instead, the value of the string-expression in the prompt-specifier shall be output (unless the input-reply is terminated by a comma -- see below). In batch mode, the input-reply shall be requested from the external source by an implementation-defined means.

If a timeout-expression is present in an input-modifier-list, then the numeric-time-expression contained therein shall be evaluated to obtain a (possibly fractional) number S of seconds. If no valid input-reply or line-input-reply has been supplied within S seconds, then an exception shall occur. A time-inquiry returns the (possibly fractional) number of seconds elapsed between the issuance of the input-prompt and the reception of the end-of-line of the last input-reply for this input-statement. This value is assigned to the numeric-variable in the time-inquiry. If no clock is provided by an implementation, then a timeout-expression shall have no effect. If a clock is provided, a time-inquiry result shall always be positive. If no clock is provided, a time-inquiry result shall be -1. The values (minimum and maximum) and resolution of both timeout expressions and time-inquiries is implementation-defined.

10.2.4.2 Assignment of Values. The assignment of a value from the input-reply to the corresponding variable shall take place as soon as an item of data in the input-reply has been validated with respect to the type of the datum and the allowable range of values for that datum.

Subscripts and substring-qualifiers in a variable-list or string-variable-list shall be evaluated after values have been

assigned to the variables preceding them (i.e., to the left of them) in the variable-list or string-variable-list.

The type of each datum in the input-reply shall correspond to the type of the variable to which it is to be assigned (i.e., numeric-constants shall be supplied as input for numeric-variables, and either string-constants or unquoted-strings shall be supplied as input for string-variables). An unquoted-string that is also a numeric-constant may be assigned to either a string-variable or a numeric-variable by an input-statement.

If the evaluation of a numeric datum causes an underflow, then its value shall be replaced by zero.

If an input-reply supplied in response to a request for input does not end with a comma, then the number of data in all the input-replies submitted shall equal the number of variables requiring values.

If the last character other than a space before the end-of-line in an input-reply is a comma, then this shall be taken to signify that further data are to be supplied. As many values as are contained in that input-reply shall be assigned to variables in the variable-list. The input-prompt (but not the string-expression of the prompt-specifier, if there is one) shall then be reissued, and execution of the program shall remain suspended until another valid input-reply has been supplied, from which further data shall be obtained.

When a line-input-statement is executed, a line-input-reply shall be requested for each string-variable in the string-variable-list in the same fashion as an input-reply is requested. That is, the value of the first line-input-reply shall be assigned to the first variable in the variable-list. If there are further variables in the variable-list, the input-prompt (but not the string-expression of the prompt-specifier, if there is one) shall then be reissued, and execution of the program shall remain suspended until a second valid line-input-reply has been supplied and assigned to the second variable in the variable-list. This process continues until a valid line-input-reply has been supplied for each variable in the variable-list. The characters of each line-input-reply, including any leading and trailing spaces, shall be concatenated to form a single string, which shall become the value of the corresponding string-variable, except that the end-of-line, which terminates a line-input-reply, shall not be included. Quotation marks within a line-input-reply are treated as actual characters. Thus, two adjacent quotation-marks are taken as two characters, not as one.

10.2.5 Exceptions

The line supplied in response to a request for an input-reply is not a syntactically correct input-reply (8102, nonfatal: request that a new input-reply be supplied).

A datum supplied as input for a numeric-variable is not a numeric-constant (8103, nonfatal: request that the current input-reply be resupplied).

There are insufficient data in an input-reply not containing a final comma (8002, nonfatal: request that the current input-reply be resupplied).

There are too many data in an input-reply or there are just enough data and the input-reply ends with a comma (8003, nonfatal: request that the current input-reply be resupplied).

The evaluation of a numeric datum causes an overflow (1007, nonfatal: request that the current input-reply be resupplied).

The assignment of a datum or a line-input-reply to a string-variable results in a string overflow (1054, nonfatal: request that the current input-reply or line-input-reply be resupplied).

The value of a numeric-time-expression is less than zero (8402, fatal).

A valid input-reply or line-input-reply has not been supplied within the number of seconds specified by a timeout-expression in an input-modifier-list (8401, fatal).

10.2.6 Remarks

This standard requires that users in the interactive mode always be given the option of resupplying erroneous input-replies; in batch mode this may be treated as a fatal exception. This standard does not require an implementation to provide facilities for correcting erroneous input-replies, though such facilities may be provided.

It is recommended that the default input-prompt consist of a question-mark followed by a single space.

This standard does not require an implementation to output (i.e., echo) an input-reply or line-input-reply.

Implementations may choose to treat underflows as exceptions (1507, nonfatal: supply zero and continue) to permit interception by exception handlers.

If an input datum is an unquoted-string, leading and trailing spaces are ignored (cf. 4.1). If it is a quoted-string, then all spaces between the quotation-marks are significant (cf. 6.1).

10.3 Output

10.3.1 General Description

The print-statement is designed for generation of tabular output in a consistent format. The set-statement with MARGIN can be used to specify the width of output-lines. The set-statement with ZONEWIDTH can be used to specify the width of print zones within a print-line. The ask-statement is used to inquire about the current MARGIN and ZONEWIDTH.

Generalizations of the print-statement are described in 10.4, 10.5, and 11.3.

10.3.2 Syntax

```
 1. print-statement      > PRINT print-list
 2. print-list           = (print-item? print-separator)*
                           print-item?
 3. print-item           = expression / tab-call
 4. tab-call             = TAB left-parenthesis index
                           right-parenthesis
 5. print-separator      = comma / semicolon
 6. set-statement        = SET set-object
 7. set-object           > (MARGIN / ZONEWIDTH) index
 8. ask-statement        > ASK ask-io-list
 9. ask-io-list          = ask-io-item (comma ask-io-item)*
10. ask-io-item          = (MARGIN / ZONEWIDTH) numeric-variable
```

A given ask-io-item shall appear at most once in an ask-statement.

10.3.3 Examples

```
 1. PRINT X
    PRINT X, Y
    PRINT X, Y, Z,
    PRINT ,,,X
    PRINT
    PRINT "X EQUALS", 10
    PRINT X; (Y+Z)/2
    PRINT TAB(10); A$; "IS DONE."
 6. SET MARGIN 120
    SET ZONEWIDTH 20
```

10.3.4 Semantics

The execution of a print-statement shall generate a string of characters and end-of-lines for transmission to an external device. This string of characters shall be determined by the successive evaluation of each print-item and print-separator in the print-list.

If an expression in a print-list invokes a function that causes a print-statement to be executed, and that print-statement transmits characters to the same device as the original print-statement, then the effect is implementation-defined.

10.3.4.1 Printing Numeric Values. Numeric-expressions shall be evaluated to produce a string of characters consisting of a leading space if the number is positive, or a leading minus-sign if the number is negative, followed by the decimal representation of the absolute value of the number and a trailing space. The possible decimal representations of a number are the same as those described for numeric-constants in 5.1 and shall be used as follows:

(1) Each implementation shall define two quantities: a significance-width d, to control the number of significant decimal digits printed in numeric representations, and an exrad-width e, to control the number of digits printed in the exrad component of a numeric representation. The value of d shall be at least 6 and the value of e shall be at least 2.

(2) Each expression whose value is exactly an integer and that can be represented with d or fewer decimal digits shall be output using the implicit point unscaled representation.

(3) All other values shall be output using either explicit-point unscaled representation or explicit-point scaled representation. Values that can be represented with d or fewer digits in the unscaled representation no less accurately than they can in the scaled representation shall be output using the unscaled representation. For example, if d = 6, then 10^(-6) is output as .000001 and 10^(-7) is output as 1.E-7.

(4) Values represented in the explicit-point unscaled representation shall be output with up to d significant decimal digits and a period; trailing zeros in the fractional part may be omitted. A number with a magnitude less than 1 shall be represented with no digits to the left of the period. This form requires up to d+3 characters counting the sign, the period, and the trailing space.

(5) Values represented in the explicit-point scaled representation shall be output in the format

significand E sign integer

where the value x of the significand is in the range $1 \leq x < 10$ and is to be represented with exactly d digits of precision, and where the exrad component has one to e digits. Trailing zeros may be omitted in the fractional part of the significand and leading zeros may be omitted from the exrad. A period shall be printed as part of the significand. This form requires up to d+e+5 characters counting the two signs, the period, the "E", and a trailing space.

10.3.4.2 Printing String Values. String-expressions shall be evaluated to generate the corresponding string of characters.

10.3.4.3 Print Separators and Tabs. The evaluation of the semicolon separator shall generate the null string (i.e., a string of zero length).

The evaluation of a tab-call or a comma separator depends upon the string of characters already generated by the current or previous print-statements. The "current line" is the (possibly empty) string of characters generated since the beginning of execution or since the last end-of-line was generated.

The "columnar position" of the current line is the print position that will be occupied by the next character output to that line. Print positions shall be numbered consecutively from the left, starting with position one. Each time a character in positions 2/0 through 7/14 of the standard character set is generated, the columnar position shall be increased by one. Each time an end-of-line is generated, the columnar position shall be reset to one. The effect of other characters on the columnar position is implementation-defined.

The "margin" is the maximum columnar position in which a character may appear. Prior to execution of a set-statement with MARGIN, the margin shall be implementation-defined, but shall not be less than the default zone width. A margin of MAXNUM shall indicate that the columnar position may be arbitrarily large.

Each print-line is divided into a fixed number of print zones in which the number of zones and the length of each zone is implementation-defined. All print zones, except possibly the last one on a line, which may be shorter, shall have the same width. The default width of a zone shall be at least d+e+6 print

positions. The zone width may be changed by the execution of a set-statement with ZONEWIDTH. ZONEWIDTH may be set to any value greater than zero, but not greater than the current margin.

The purpose of the tab-call is to set the columnar position of the current line to the specified value prior to printing the next print-item. More precisely, the argument of the tab-call shall be evaluated and rounded to the nearest integer n. If n is less than one, an exception shall occur. If n is greater than the margin m, then n shall be reduced by an integral multiple of m so that it is in the range $1 \leq n \leq m$ (i.e., n shall be set equal to MOD(n-1,m) + 1).

If the columnar position of the current line is less than or equal to n, then spaces shall be generated, if necessary, to set the columnar position to n; if the columnar position of the current line is greater than n, then an end-of-line shall be generated followed by n-1 spaces to set the columnar position of the new current line to n.

The evaluation of the comma print-separator depends upon the columnar position. If this position is neither in the last print zone on a line nor beyond the margin, then one or more spaces shall be generated to set the columnar position to the beginning of the next print zone on the line. If the initial columnar position is in the last print zone on a line, then an end-of-line shall be generated. Finally, if the initial columnar position is beyond the margin (as it would be if evaluation of the last print-item exactly filled the line), then an end-of-line shall be generated.

10.3.4.4 Overlength Output Lines. Whenever the columnar position is greater than one and the generation of the next print-item would cause a character to appear beyond the margin, then an end-of-line shall be generated prior to the characters generated by that print-item.

During the generation of a print-item, whenever that generation would cause a character to appear beyond the margin, an end-of-line shall be generated prior to that character, resetting the columnar position to one.

10.3.4.5 End of Print-List. When evaluation of a print-list is completed, if that print-list did not end with a print-separator, then a final end-of-line shall be generated; otherwise, no such final end-of-line shall be generated.

A completely empty print-list shall generate an end-of-line, thereby completing the current line of output. If this line contained no characters, then a blank line shall result.

10.3.4.6 Setting the Margin. Execution of a set-statement with a MARGIN shall cause its index to be evaluated and to become the new margin. The change in the margin shall take effect immediately, even if a line of output is partially filled. The set-statement with a MARGIN affects only unformatted output.

10.3.4.7 Setting the Zone Width. Execution of a set-statement with a ZONEWIDTH shall cause its index to be evaluated and to become the new zone width. The change in the zone width shall take effect immediately, even if a line of output is partially filled. The set-statement with a ZONEWIDTH affects only unformatted output.

10.3.4.8 Ask-Statement. Execution of an ask-statement shall cause the variables in the ask-io-list to be assigned values corresponding to the current margin, if MARGIN is present, or current zonewidth, if ZONEWIDTH is present. If the columnar position may be arbitrarily large, then the value MAXNUM shall be returned to the numeric-variable in the ask-statement with MARGIN.

10.3.5 Exceptions

The value of the index in a tab-call is less than one (4005, nonfatal: supply one and continue).

The value of the index in a set-statement with a MARGIN is less than the current zonewidth (4006, fatal).

The value of the index in a set-statement with a ZONEWIDTH is less than one or greater than the current margin (4007, fatal).

10.3.6 Remarks

The character string generated by printing the value of a numeric-expression contains a single trailing space. If the generation of that space would cause the columnar position to exceed the margin by more than one, then implementations may choose not to generate that space, thereby allowing the number to be printed in the final print zone on a line.

Implementations may choose to use a lowercase "e" in printing numerical values using the explicit-point scaled representation.

The print-separator following a tab-call is significant in the same manner that it is significant following an expression.

10.4 Formatted Output

10.4.1 General Description

A print-statement may control the format of output by specifying an image to which that output must conform. The image is specified either within the print-statement or in a separate image-line.

10.4.2 Syntax

```
 1. print-statement       > PRINT formatted-print-list
 2. formatted-print-list = USING image (colon output-list)?
 3. image                = line-number / string-expression
 4. output-list          = expression (comma expression)*
                           semicolon?
 5. image-line           = line-number IMAGE colon
                           format-string end-of-line
 6. format-string        = literal-string
                           (format-item literal-string)*
 7. literal-string       = literal-item*
 8. literal-item         = letter / digit /
                           apostrophe / colon / equals-sign /
                           exclamation-point /
                           left-parenthesis / question-mark /
                           right-parenthesis / semicolon /
                           slant / space / underline
 9. format-item          = (justifier? floating-characters
                           (i-format-item / f-format-item /
                           e-format-item)) / justifier
10. justifier            = greater-than-sign / less-than-sign
11. floating-characters  = (plus-sign* / minus-sign*)
                           dollar-sign? /
                           dollar-sign* (plus-sign /
                           minus-sign)?
12. i-format-item        = digit-place digit-place* (comma
                           digit-place digit-place*)*
13. digit-place          = asterisk / number-sign / percent-sign
14. f-format-item        = period number-sign number-sign* /
                           i-format-item period number-sign*
15. e-format-item        = (i-format-item / f-format-item)
                           circumflex-accent circumflex-accent
                           circumflex-accent circumflex-accent*
```

An image that is a line-number shall refer to an image-line in the same program-unit. Any leading spaces following the colon in an image-line are part of the format-string.

All digit-places in an i-format-item shall be the same character (i.e., all shall be number-signs, all shall be percent-signs, or all shall be asterisks).

10.4.3 Examples

```
  10 LET sum = 20
  20 PRINT USING "The answer is ###.#": sum

produces          "The answer is  20.0".

  30 PRINT USING 40: 342, 42.021
  40 IMAGE :RATE OF LOSS #### EQUALS ####.## POUNDS

produces    "RATE OF LOSS  342 EQUALS   42.02 POUNDS".

  10 LET A$ = "<##### ####.#### ####.####^^^^"
  20 PRINT USING A$: 1, 1, 1

produces       "     1    1.0000 1000.0000E-03".

  60 PRINT USING 70: "ONE", "TWO", "THREE"
  70 IMAGE :Z<####>#### ########Z

produces    "ZONE    TWO  THREE  Z".

  80 LET A$ = "Pay $**.## on ### %% 19%%"
  90 PRINT USING A$: 1, "May", 2, 83

produces       "Pay $*1.00 on May 02 1983".

  10 PRINT USING "<%%.## >---$##.## $$$+***": 3.1, -1234.567, 2

produces          "003.10  -$1234.57   $+**2".

  10 PRINT USING "<$$$$.## $$$$.###^^^^^": -.02, -.02

produces          "   $-.02   $-.200E-001".

  10 PRINT USING "$***,***.##": 1234.7777

produces          "$**1,234.78".
```

10.4.4 Semantics

A print-statement with a formatted-print-list identifies a format-string to be used to control the output generated by the evaluation of the output-list. If the image is specified via a

line-number, then the format-string is contained in the image-line with the indicated line-number; otherwise, it is the value of the string-expression.

10.4.4.1 Format-String Analysis. The selected format-string shall be analyzed as a number of format-items separated by possibly zero-length literal-strings. Format-items shall be found within the format-string by scanning the latter from left to right. A search shall be made for the first character that is the start of a syntactically correct format-item, and the longest such format-item starting at that character identified. The scan for format-items shall continue in this way up to the end of the format-string, the search for the start of each new format-item beginning at the character immediately beyond the previously identified format-item. Corresponding to each format-item shall be an output field whose length equals the number of characters in the format-item (including the justifier, floating-characters, digit-places, commas, period, number-signs, and circumflex-accents). Characters that are not part of any format-item shall be literal-items.

Format-strings that are defined in image-lines shall be interpreted as ending with the last character in the line which is not a space or end-of-line.

10.4.4.2 Literal-Strings and Output Fields. A sequence of values to be output shall be generated by evaluating each expression in the output-list in sequence. As each value is generated, the literal-string preceding the next format-item in the format-string shall first be copied unchanged into the string of characters being generated. Then a number of characters equal to the length of the output field determined by that format-item shall be generated according to 10.4.4.3, if the expression is numeric, or according to 10.4.4.4, if the expression is a string.

10.4.4.3 Formatted Numeric Output. Numeric values shall be rounded and represented in a manner corresponding to the format-item used. If a justifier is present in the format-item, it shall be replaced by the character immediately to its right. If, however, the character to its right is a period, or if there is no character to its right, then the justifier shall be replaced by a number-sign.

First, a representation for the magnitude of the value shall be generated.

For an i-format-item, the value shall be rounded to the nearest integer and represented using implicit point unscaled notation.

For an f-format-item, the value shall be represented using explicit point unscaled notation, rounding the representation or extending it on the right with zeros in accordance with the number of number-signs following the period in the format-item.

For both i-format-items and f-format-items, leading zeros to the left of the implicit or explicit decimal point shall not be generated, unless this results in no digits being generated. In that case, the character "0" shall be generated immediately to the left of the explicit or implicit decimal point. After this, if there remain unfilled digit-places, then leading zeros shall be generated in the integer or to the left of the period when a percent-sign is used as a digit-place, leading asterisks when an asterisk is used as a digit-place, and leading spaces when a number-sign is used as a digit-place, such that the number of characters to the left of the implicit or explicit decimal point is equal to the number of digit-places in the format-item.

For an e-format-item, the value shall be represented using explicit or implicit point scaled notation, corresponding to the use of an f-format-item or i-format-item, respectively, within the e-format-item. The significand for nonzero values shall be scaled by powers of ten such that the leftmost digit-place or number-sign position is occupied by a nonzero digit. In all other respects, the significand shall be generated according to the above rules for i-format-items and f-format-items. The number of circumflex-accents in an e-format-item shall determine the number of characters in the exrad. The first of these characters shall be the letter E, the next a mandatory sign, and the remaining characters the representation of the magnitude of the exrad, with leading zeros being generated so that the number of characters in the exrad equals the number of circumflex-accents in the format-item. If the exponent is zero, the mandatory sign is positive; the exponent of zero is zero.

Second, commas shall be inserted in this numeric representation wherever a comma occurs in the format-item, provided at least one digit has been generated to the left of the point of insertion; if no digit has been generated to the left of this point, then an asterisk shall be inserted if the digit-place immediately to the left is an asterisk, and a space inserted if the digit-place immediately to the left is a number-sign.

Third, leading characters composed of sign, dollar-sign, and space shall be generated as follows:

Floating-characters		Leading Characters Generated	
First*	Last	Non-negative	Negative
-	$	" $"	"-$"
$	-	"$ "	"$-"
-	none	" "	"-"
+	$	"+$"	"-$"
$	+	"$+"	"$-"
+	none	"+"	"-"
$	none	"$"	"$-"
none	none	""	"-"

* may be several occurrences

Finally, the representation of the numeric value so generated shall be extended by spaces on the left so that its length equals that of the format-item. This has the effect of right-justifying a numeric-representation in an output field.

10.4.4.4 Formatted String Output. A string value may be output using any type of format-item. The string shall be extended by spaces so that its length equals that of the format-item. These spaces shall be added on the left (for right-justification) if the format-item begins with a greater-than-sign, on the right (for left-justification) if it begins with a less-than-sign, and equally on either side (for centering) otherwise; if the number of spaces required in the last case is odd, the extra space shall be added on the right.

10.4.4.5 Formatted Output Completion. If the number of values to be output exceeds the number of format-items in the format-string, an end-of-line shall be generated each time the end of the format-string is reached and the format-string reused for the remaining expressions. If format-items remain in the format-string after all values have been output, then the next literal-string, if any, shall be output. Generation of characters is always terminated beginning at the first unused format-item.

Finally, an end-of-line shall be generated after all other character generation is completed, unless the output-list ends with a semicolon, in which case no such end-of-line shall be generated.

The current margin shall not affect the output; in particular, no end-of-line shall be generated upon formatted output just because the margin is exceeded.

If the execution of a program reaches an image-line, it shall proceed to the next line with no further effect.

10.4.5 Exceptions

An invalid format-string is specified in a formatted-print-list (8201, fatal).

A formatted-print-list contains an output-list, but there is no format-item in the format-string (8202, fatal).

An output string, whether generated from a string-expression or a numeric-expression, is longer than its corresponding format-item (8203, nonfatal: fill the output field with asterisks, report the unformatted representation of the value on the next line, and continue printing on the following line in a position identical to the position that would have resulted if no exception had occurred).

The exrad for numeric output exceeds the space allocated by circumflex-accents in a format-item (8204, nonfatal: fill the output field with asterisks, report the unformatted representation of the value, and continue).

10.4.6 Remarks

Since format-strings may be evaluated dynamically, errors in them (even if occurring in an image-line and therefore statically determined) may be treated as exceptions.

Implementations may choose to use a lowercase "e" in printing numerical values using the explicit-point scaled representation.

The integer part of a number generated with an i-format-item or f-format-item may validly contain more digits than there are digit-places in the format-item, as long as the floating-characters provide sufficient room.

Negative numeric values always generate a minus-sign. The corresponding format-item must provide room for this minus-sign with floating-characters, since digit-places are completely filled by digits, or by leading spaces, zeros, or asterisks. In particular, a format-item with no floating-characters, or with

only a single dollar-sign as a floating-character, will cause exception 8203 if an attempt is made to fill that field with a negative value.

10.5 Array Input and Output

10.5.1 General Description

Statements are provided that enable entire arrays to be input or output. These statements generalize the input and output statements that manipulate single values (cf. 10.1 to 10.4).

10.5.2 Syntax

```
 1. array-read-statement        > MAT READ (missing-recovery
                                  colon)? redim-array-list
 2. redim-array-list            = redim-array
                                  (comma redim-array)*
 3. redim-array                 = array-name redim?
 4. array-input-statement       > MAT INPUT input-modifier-list?
                                  (redim-array-list /
                                  variable-length-vector)
 5. variable-length-vector      = array-name left-parenthesis
                                  question-mark right-parenthesis
 6. array-line-input-statement  > MAT LINE INPUT
                                  input-modifier-list?
                                  redim-string-array-list
 7. redim-string-array-list     = redim-string-array
                                  (comma redim-string-array)*
 8. redim-string-array          = string-array redim?
 9. array-print-statement       > MAT PRINT (array-print-list /
                                  (USING image colon
                                  array-output-list))
10. array-print-list            = array-name (print-separator
                                  array-name)* print-separator?
11. array-output-list           = array-name (comma array-name)*
                                  semicolon?
```

A redim and the array in its redim-array shall have the same number of dimensions.

A variable-length-vector shall be one-dimensional.

10.5.3 Examples

```
1. MAT READ A
   MAT READ A(M,N), B
4. MAT INPUT A$(3,4)
   MAT INPUT X(?)
   MAT INPUT PROMPT "Enter data:": X(?)
```

8. MAT LINE INPUT A$
9. MAT PRINT A; B, C;

10.5.4 Semantics

10.5.4.1 The Array-Read-Statement. Execution of an array-read-statement shall cause arrays in the redim-array-list to be assigned, in order, values from the data sequence created by data-statements in the program-unit containing that statement. Values shall be assigned to all elements in each array in "row major order" (i.e., the last subscript varying most rapidly, then the next to last subscript, if any, and so on) with each successive value being obtained from the datum in the data sequence indicated by the pointer for the sequence and the pointer being advanced beyond that datum.

The type of each datum in the data sequence shall correspond to the type of the array-element to which it is to be assigned (cf. 10.1).

If a redim is present then dynamic redimensioning shall take place before values are assigned to the redimensioned array. The redimensioning shall be done according to the rules for bounds in array-declarations. The values of the indices shall be used as the new lower and upper bounds for the array. If an exception occurs when attempting to redimension an array, it shall retain its old dimensions. Redims in the redim-array-list shall be evaluated after values have been assigned to the arrays preceding them (i.e., to the left of them) in the redim-array-list.

The handling of insufficient data with or without a missing-recovery shall work as described in 10.1.

If the evaluation of a numeric datum causes an underflow, then its value shall be replaced by zero.

10.5.4.2 The Array-Input-Statement. Execution of an array-input-statement shall cause execution of the program to be suspended until a valid input-reply, as specified below, has been supplied. An array-input-statement shall cause arrays in the redim-array-list to be assigned, in order, values from the input-reply. Values shall be assigned to all elements in each array in row major order.

In the interactive mode, the user of the program shall be informed of the need to supply data by the output of an input-prompt. The prompt is identical to that of the input-prompt of the input-statement.

The input-modifier-list, if present, shall work as described in 10.2.

The type of each datum in the input-reply shall correspond to the type of the array-element to which it is to be assigned.

If a redim is present then dynamic redimensioning shall take place as described above for the array-read-statement before values are assigned to the redimensioned array. Redims in the redim-array-list shall be evaluated after values have been assigned to the redim-arrays preceding them (i.e., to the left of them) in the redim-array-list.

If the recovery procedure for an input exception causes input data to be resupplied to an array that was redimensioned after the original assignment of data to it, but before the exception occurred, the effect is implementation-defined.

Data in response to a request for array input need not be supplied in a single input-reply. If the array-list has not been completely supplied with data and the input-reply contains a final comma, then the input-prompt shall be issued and a further input-reply shall be requested to obtain more data.

If the evaluation of a numeric datum causes an underflow, then its value shall be replaced by zero.

10.5.4.3 Input of Variable-Length Vectors. If a variable-length-vector occurs in an array-input-statement, then as many data as are present in the input-reply (or sequence of input-replies up to and including the first that does not end with a comma) shall be supplied as input for that vector. Assignment of data shall begin with the current lower bound for the vector. After assignment, the vector shall be redimensioned dynamically by setting the upper bound for its subscript equal to the subscript of the element receiving the last datum. The number of data values assigned to the variable-length-vector shall not exceed the original size for that vector as specified in its array-declaration.

The type of each datum in the input-reply shall correspond to the type of the array.

10.5.4.4 The Array-Line-Input-Statement. When an array-line-input-statement is executed, a line-input-reply shall be requested for each element of each string-array in the string-array-list in the same fashion that an input reply is requested and shall assign the entire contents of successive

line-input-replies (excluding their end-of-lines) in row major order to elements of the string-arrays in the string-array-list. The number of line-input-replies requested shall equal the number of elements requiring values.

In the interactive mode, the user of the program shall be informed of the need to supply data by the output of an input-prompt.

The input-modifier-list, if present, shall work as described in 10.2.

If a redim is present, then dynamic redimensioning shall take place as described above for the array-read-statement before values are assigned to the redimensioned array. Redims in the redim-string-array-list shall be evaluated after values have been assigned to the redim-string-arrays preceding them (i.e., to the left of them) in the redim-string-array-list.

10.5.4.5 The Array-Print-Statement. Execution of an array-print-statement shall cause the values of all elements in all arrays in the array-print-list to be printed. An end-of-line shall be generated prior to any characters generated by an array-print-statement if the current line of output is nonempty.

For an array-print-statement with an array-print-list, the characters generated for transmission to an external device by the printing of a two-dimensional array are almost precisely those that would be generated if the elements in that array had been listed, row by row, in the print-list of a print-statement, separated by the separator which follows the array-name in the array-print-list (or separated by a comma if no separator follows the array name). The only additional characters generated shall be an end-of-line each time a row of the array has been printed (if such an end-of-line has not already been generated). A three-dimensional array shall be printed like a series of two-dimensional arrays, one for each value of the first subscript, with an extra end-of-line generated between each value of the first subscript. When a one-dimensional array is printed, it shall be treated like a row-vector, and printed as if it were a 1 x N array.

Finally, an extra end-of-line shall be generated between the output for successive arrays in an array-print-list.

For an array-print-statement with an array-output-list, the characters generated for transmission to an external device are exactly those that would be generated if the elements of each

array had been listed array-by-array, in row major order, in the output-list of a print-statement, using the same image as that in the array-print-statement. No additional end-of-lines shall be generated. As with a print-statement using an image, if there is no trailing semicolon in the array-output-list, a final end-of-line shall be generated after all other output from the array-print-statement. If there is such a semicolon, then this final end-of-line shall not be generated.

10.5.5 Exceptions

The redim-array-list in an array-read-statement requires more data than are present in the remainder of the data sequence and no missing-recovery has been specified (8001, fatal).

An attempt is made to assign a value to an element of a numeric-array from a datum in the data sequence that is not a numeric-constant (8101, fatal).

The assignment of a datum during execution of an array-read-statement results in a string overflow (1053, fatal).

The evaluation of a numeric datum in a data-list causes an overflow (1006, fatal).

The line supplied in response to a request for array input is not a syntactically correct input-reply (8102, nonfatal: request that a new input-reply be supplied).

A datum supplied as input for a numeric-array is not a numeric-constant (8103, nonfatal: request that the current input-reply be resupplied).

There are insufficient data in an input-reply not containing a final comma (8002, nonfatal: request that the current input-reply be resupplied).

There are too many data in an input-reply or there are just enough data and the input-reply ends with a comma (8003, nonfatal: request that the current input-reply be resupplied).

The evaluation of a numeric datum in an input-reply causes an overflow (1007, nonfatal: request that the current input-reply be resupplied).

The assignment of a string datum during execution of an array-input-statement or an array-line-input-statement causes a

string overflow (1054, nonfatal: request that the current input-reply or line-input-reply be resupplied).

The total number of elements required for a redimensioned array exceeds the number of elements reserved by the array's original dimensions (5001, fatal).

The first index in a redim-bounds is greater than the second index (6005, fatal).

A redim-bounds consists of a single index that is less than the default lower bound in effect (6005, fatal).

A valid input-reply or line-input-reply has not been supplied within the number of seconds specified by a timeout-expression in an input-modifier-list (8401, fatal).

The value of a numeric-expression used as a time-expression is less than zero (8402, fatal).

An invalid format-string is specified in an array-print-statement (8201, fatal).

An array-print-statement contains an array-output-list, but there is no format-item in the format-string (8202, fatal).

10.5.6 Remarks

This standard does not require an implementation to output (i.e., echo) the input-reply or line-input-reply.

This standard does not require an implementation to provide facilities for correcting erroneous input-replies, though such facilities may be provided.

Implementations may choose to treat underflows as exceptions (1507, nonfatal: supply zero and continue) to permit interception by exception handlers.

11. Files

Files are organized collections of data external to BASIC programs. They provide the user with a means of saving data developed during execution of a program and then retrieving and modifying that data during subsequent executions of BASIC programs. The process by which external data is transferred to or from a program is called input or output, respectively. An implementation-defined means shall be provided for the creation, preservation, and retrieval of files. Input and output operations to these files shall perform as specified in this section.

This section describes the logical appearance of files and devices to a BASIC program. In some cases, these attributes may reflect physical characteristics, but in general this standard makes no presumptions concerning the physical representation or organization of files or devices.

There are four kinds of file-organization: sequential, stream, relative, and keyed. Sequential and keyed files are sequences of records. A relative file is a sequence of record areas, each of which may or may not contain a record. A stream file is a sequence of values.

There are three kinds of record-type: display, internal, and native. A display record is a sequence of characters. An internal record is a sequence of typed values. A native record is a sequence of fields, as described by a program-specified template. Display records provide for the exchange of data between systems employing different internal representations for numeric and string values, and also manipulate data in human-readable form. Internal records provide for efficient manipulation of data within a single system. Native records provide for the exchange of data among different language processors within a single system.

There are three modules provided for file capabilities, based on which combinations of file-organization and record-type are supported. The core module contains sequential display files, sequential internal files, and stream internal files. The enhanced internal module contains relative internal and keyed internal files. The enhanced native module contains sequential native, relative native, and keyed native files. All other combinations of file-organization and record-type are implementation-defined. The distinction between modules is also reflected in the arrangement of the productions for the syntax. Within each subsection, the syntax rules for the core module are

presented first, followed by the additional syntax productions that pertain to the enhanced files modules. Some of the enhanced productions apply only to the enhanced-native module; these are preceded by an "N".

The meaning of certain terms used throughout this section is as follows. A "file element" is an entity, a sequence of which constitutes a file. Thus, for keyed and sequential files, a file element is a record; for relative files, it is a record-area; for stream files, it is a value. Associated with each file during execution is a "file pointer," which either uniquely identifies a particular file element upon completion of any statement or points to the end of file. If the pointer is at the beginning of the file, then it identifies the first file element, if any. If a file is an empty sequence, then the beginning and end of file are the same, and the pointer identifies this location. Whenever reference is made to the "next" file element, it is understood that if none such exists, the end of file is substituted. For sequential, stream, and keyed files, the "end of file" is the location immediately following the last file element. For relative files, the "end of file" immediately follows the last existing record, and thus identifies an empty record-area.

There are five statements that operate on the file as a whole and are thus called "file operations": OPEN, CLOSE, ERASE, SET, and ASK. There are seven statements that apply to individual file elements and are known as "record operations": INPUT, PRINT, READ, WRITE, REWRITE, DELETE, and SET with pointer-control, including the variations using MAT and LINE. References to "INPUT operations," "WRITE operations," and so forth should be understood to include any of the statements using the keyword in question, e.g., "WRITE operations" includes WRITE and MAT WRITE. The seven record operations can affect (1) data within a file, (2) variables within the program, and (3) the file pointer. PRINT, WRITE, REWRITE, and DELETE affect file data and the pointer. READ and INPUT affect program variables and the pointer. SET with pointer-items obviously affects only the pointer.

Not all input and output is to or from a file, as defined above. An implementation may allow file processing statements to apply as well to devices, such as a terminal, a line printer, or a communications line.

When the term "file" is used throughout Section 11, it should generally be understood to mean any source or destination of external data (i.e., either a true file or a device). In certain contexts in which it is necessary to distinguish between

the two, the terms "true file" and "device" will be used for emphasis.

Devices differ from files in the following ways:

(1) It is implementation-defined whether data written to any given device is stored there and may later be retrieved by input operations (see 11.1).

(2) It is implementation-defined whether a given device is erasable (see 11.1).

(3) RELATIVE and KEYED file-organizations are not allowed for devices (see 11.1).

(4) A device need not support all access-modes (see 11.1).

(5) A device need not support the minimum record-size of 132 (see 11.1). However, the implementation shall document the minimum record-size for each device supported.

(6) It is implementation-defined whether a given device has record-setter capability (see 11.2).

(7) It is implementation-defined what conditions cause the data-found condition to be set true or false for a given device (see 11.2).

(8) For interactive terminal devices only, the semantics of the input-control-items prompt-specifier, timeout-expression, and time-inquiry shall be supported. The implementation shall define which devices, if any, are interactive terminal devices. The effect of these input-control-items on other devices and on true files is implementation-defined (see 11.4).

(9) It is implementation-defined whether the following conditions are treated either as fatal exceptions, as defined in Section 11, or as nonfatal exceptions, as defined in Section 10 (in which case the recovery procedure is applied), when these conditions occur within INPUT operations on a device (see 11.4).

Section 11 EXTYPE	Section 10 EXTYPE	Condition
8105	8102	Syntax error in input-reply
8101	8103	Datum for numeric-variable not a numeric-constant
8012	8002	Too few data in input-reply

8013	8003	Too many data in input-reply
1008	1007	Numeric overflow on input
1105	1054	String overflow on input

Tables 2 through 4 provide an overview of the various file facilities. For the full specification, see 11.1 through 11.5.

Table 2 illustrates which combinations of record operations and record-setters are legal under a given file-organization; thus, the organization is defined by the capabilities for record manipulation it allows. Combinations of operations and record-setters that do not appear in the table (e.g., INPUT with KEY) are syntax errors.

Table 2. File-Organization vs. Operations and Record-Setters

Operation record-setter	File Organization SEQUENTIAL	STREAM	RELATIVE	KEYED
INPUT				
absent	OK	ID 2	ID 2	ID 2
NEXT	OK	ID 2	ID 2	ID 2
BEGIN	OK	ID 2	ID 2	ID 2
END	OK 4	ID 2	ID 2	ID 2
SAME	OK	ID 2	ID 2	ID 2
PRINT				
absent	OK	ID 2	ID 2	ID 2
NEXT	OK	ID 2	ID 2	ID 2
BEGIN	OK	ID 2	ID 2	ID 2
END	OK	ID 2	ID 2	ID 2
SAME	OK	ID 2	ID 2	ID 2
READ				
absent	OK	OK	OK	OK
NEXT	OK	OK	OK	OK
BEGIN	OK	OK	OK	OK
END	OK 4	OK 4	OK 4	OK 4
SAME	OK	OK	OK	OK
RECORD	EX 1	EX 1	OK	EX 1
KEY	EX 1	EX 1	EX 1	OK

WRITE				
absent	OK	OK	OK	EX 3
NEXT	OK	OK	OK	EX 3
BEGIN	OK	OK	OK	EX 3
END	OK	OK	OK	EX 3
SAME	OK	OK	OK	EX 3
RECORD	EX 1	EX 1	OK	EX 1,3
KEY (exact)	EX 1	EX 1	EX 1	OK
REWRITE and DELETE				
absent	ID 5	ID 5	OK	OK
NEXT	ID 5	ID 5	OK	OK
BEGIN	ID 5	ID 5	OK	OK
END	ID 5	ID 5	OK 4	OK 4
SAME	ID 5	ID 5	OK	OK
RECORD	ID 5	ID 5	OK	EX 1
KEY	ID 5	ID 5	EX 1	OK
SET with pointer-items				
absent	OK	OK	OK	OK
NEXT	OK	OK	OK	OK
BEGIN	OK	OK	OK	OK
END	OK	OK	OK	OK
SAME	OK	OK	OK	OK
RECORD	EX 1	EX 1	OK	EX 1
KEY	EX 1	EX 1	EX 1	OK

LEGEND: OK = Semantics defined by standard; EX = Exception; ID = implementation-defined.

NOTES:

(1) RECORD is valid only with RELATIVE files and KEY with KEYED files.

(2) INPUT and PRINT are defined only for record-type DISPLAY, and DISPLAY is defined only for SEQUENTIAL.

(3) WRITE to a KEYED file shall specify an exact key search.

(4) END implies that data-found will be false.

(5) REWRITE and DELETE are implementation-defined for file-organizations other than RELATIVE and KEYED.

Table 3 illustrates which control features are allowed syntactically with the various operations.

Table 3. Record Operations vs. Controls

	Controls					
	record-setter		io-recovery		interpretation	
	core-setter	any setter	missing	not missing	image	template
INPUT	A		A			
PRINT	A			A	A	
READ		A	A			A
WRITE		A		A		A
REWRITE		A	A			A
DELETE		A	A			
SET **		A	A	A		

LEGEND: A = allowed; core-setter = NEXT / BEGIN / END / SAME; ** = with pointer-items.

Table 4 illustrates which combinations of file-organization and record-type are defined by the standard, and if so at which level.

Table 4. File-Organization vs. Record-type

	Record-type		
Organization	DISPLAY	INTERNAL	NATIVE
SEQUENTIAL	core	core	enhanced-native
STREAM		core	
RELATIVE		enhanced-internal	enhanced-native
KEYED		enhanced-internal	enhanced-native

11.1 File Operations

11.1.1 General Description

There are four statements that affect a file as an entity. The open-statement makes a file accessible to the program, establishing the connection between the file and the program. Since the format for identifying files may vary with the operating system, it is assumed only that with each file is associated a string of characters, called its name, which identifies the file to the operating system. A file is identified within a program by the number of a channel through which it is accessed. The close-statement terminates the accessibility effected by the open-statement. The erase-statement deletes all or part of the data within a true file, but may have no effect on a device. The ask-statement is used to inquire about the current status of the file.

11.1.2 Syntax

Core productions:

```
 1. open-statement        = OPEN channel-setter NAME file-name
                            file-attribute-list
 2. channel-setter        = channel-expression colon
 3. channel-expression    = number-sign index
 4. file-name             = string-expression
 5. file-attribute-list   = (comma file-attribute)*
 6. file-attribute        > core-file-attribute
 7. core-file-attribute   = access-mode / file-organization /
                            record-type / record-size
 8. access-mode           = ACCESS (INPUT / OUTPUT / OUTIN /
                            string-expression)
 9. file-organization     = ORGANIZATION
                            (file-organization-value /
                            string-expression)
10. file-organization-value > core-file-org-value
11. core-file-org-value   = SEQUENTIAL / STREAM
12. record-type           = RECTYPE (record-type-value /
                            string-expression)
13. record-type-value     > core-record-type-value
14. core-record-type-value = DISPLAY / INTERNAL
15. record-size           = RECSIZE (VARIABLE /
                            string-expression) (LENGTH index)?
16. close-statement       = CLOSE channel-expression
17. erase-statement       = ERASE REST? channel-expression
18. ask-statement         > ASK channel-setter ask-item-list
19. ask-item-list         = ask-item (comma ask-item)*
```

```
  20. ask-item              = ask-attribute-name variable
                              variable*
  21. ask-attribute-name    > core-attribute-name
  22. core-attribute-name   = ACCESS / DATUM / ERASABLE /
                              FILETYPE / MARGIN / NAME /
                              ORGANIZATION / POINTER / RECSIZE /
                              RECTYPE / SETTER / ZONEWIDTH
```

Enhanced Files productions:

```
  23. file-organization-value > enhanced-file-org-value
  24. enhanced-file-org-value = RELATIVE / KEYED
 N25. record-type-value       > enhanced-record-type-value
 N26. enhanced-record-type-value = NATIVE
  27. file-attribute          > enhanced-file-attribute
  28. enhanced-file-attribute = collate-sequence
  29. collate-sequence        = COLLATE (STANDARD / NATIVE /
                                string-expression)
  30. ask-attribute-name      > enhanced-attribute-name
  31. enhanced-attribute-name = RECORD / KEY / COLLATE
```

A given file-attribute shall appear at most once in a file-attribute-list.

A given ask-attribute-name shall appear at most once in an ask-item-list.

The number and types of variables in an ask-item shall agree with Table 5 in 11.1.4.9.

11.1.3 Examples

```
   1. OPEN #3: NAME "myfile"
      OPEN #N: NAME A$, ACCESS OUTIN, ORGANIZATION KEYED,
               RECTYPE INTERNAL, RECSIZE VARIABLE LENGTH N
      OPEN #N+1: NAME "MY" & F$, ORGANIZATION ORG$
  16. CLOSE #N
  17. ERASE #3
      ERASE REST #4
  18. ASK #3: ACCESS AC$, DATUM DT$, NAME NM$, ORGANIZATION ORG$,
              POINTER P$, RECSIZE RS$ NUMCHARS, RECTYPE RT$
      ASK #N: KEY K$
```

11.1.4 Semantics.

Files are accessed through channels to which they may be assigned during execution of a program-unit. A channel is a logical path through which external data may be transferred to or

from a BASIC program. Within a program-unit, a channel is identified by a channel number local to that program-unit. The channel number is an integer from 0 up to and including some implementation-defined maximum. This maximum shall be at least 99. A file, identified by its file-name, is open if it is currently assigned to a channel and closed otherwise. A channel is active if it currently has some file assigned to it and inactive otherwise. At the initiation of execution of a program, all channels except channel zero shall be inactive. Channel zero shall always be active. Execution of the open-, close-, or erase-statement (see below) for channel zero shall cause a nonfatal exception.

Input and output from and to channel zero shall have the same source and destination as input-statements and print-statements that do not contain channel-expressions. Channel zero shall behave as a device with the file-attributes sequential, display, and outin, and without record-setter or erase capability.

11.1.4.1 Open-Statement. The open-statement makes the file identified by the file-name accessible to the program through the channel number specified in the channel-expression. It is implementation-defined whether file names differing only in the case of the letters (upper or lower) denote the same file or different files. Following a successful open-statement, the associated channel shall be active and the file open. An attempt to open a file on a channel that is already active causes an exception. The effect of attempting to open a file that is already open is implementation-defined. The number of channels other than channel zero that may be active simultaneously shall be at least one for implementations conforming to the core, and at least two for implementations conforming to the enhanced file module.

After a successful open, a true file shall be accessible in accordance with the associated file-attributes, whether explicitly specified or in effect by default. This accessibility consists of the ability to perform certain operations and manipulate the file pointer in certain ways. See Section 11 for an overview of which statements are allowed under which attributes. If an attempt is made to OPEN a file that cannot be made accessible with the requested attributes (i.e., if not all the associated operations can be successfully executed for this file), then an exception results.

For a device, a successful open guarantees that, with two exceptions, all the file processing statements will have the same

effect as for a true file. In particular, on output, the same data will be generated, and on input, values and characters will be interpreted and assigned to variables in the same way. A device, however, might not support the semantics associated with the record-setter (cf. 11.2) or the erase-statement (cf. 11.1.4.8). The ask-statement may be used to determine whether a particular device supports these capabilities.

If a file is opened successfully with a given file-organization, record-type, and record-size, then closed, and then opened at a later time with a different value for one of these file-attributes, then it is implementation-defined whether the file is thus accessible. Also, for files with record-type INTERNAL or NATIVE, if a different ARITHMETIC option is in effect for the two executions, it is implementation-defined whether the file is thus accessible. Conversely, if a true file is re-opened at a later time with the same values for the file-attributes mentioned and the same collate-sequence, and, for files with record-type INTERNAL and NATIVE, the same ARITHMETIC option is in effect, and the user has employed the implementation-defined means to preserve the file unchanged in the interim, then the file shall be accessible and the contents of the file faithfully preserved. Devices are not required to preserve data. In the foregoing, "same ARITHMETIC option" refers to DECIMAL or NATIVE or FIXED (cf. 15.1), not to the default specification in the FIXED option. If a KEYED file is reopened with a different collate-sequence, an exception results.

If a file with record-type INTERNAL or NATIVE opened in one program-unit is accessed by another program-unit with a different ARITHMETIC option, the results are implementation-defined.

Implementations must provide true files for which all access-modes are available. Implementations may also support true files for which some access-modes are not available. A device need not support all access-modes.

Implementations conforming only to the core module shall accept and process three combinations of file-organization-value and record-type-value, namely, sequential and display, sequential and internal, and stream and internal. The effect of any other combination is implementation-defined.

Implementations conforming to the enhanced-internal module shall accept and process, in addition to those of the core module, relative and internal, and keyed and internal.

Implementations conforming to the enhanced-native module shall accept and process, in addition to those of the core module, sequential and native, relative and native, and keyed and native.

When a string-expression is used as an attribute value, its value must be one of the associated keywords for that attribute. Upper- and lower-case-letters shall be treated as equivalent within such string values. Implementations may define additional file attribute values.

11.1.4.2 Access-Mode. An access-mode specifies the direction in which data may be transferred from and to a file, either by one of the keywords INPUT, OUTPUT, or OUTIN, or by a string-expression whose value is one of these keywords.

If the access-mode is INPUT, then it shall be possible to read data from the file, but not to change the file. In particular, READ, SET with pointer-items, and INPUT statements (including variations with MAT and LINE) are allowed, but not PRINT, WRITE, REWRITE,or DELETE.

If the access-mode is OUTPUT, then it shall be possible to add new data to the file, but not to change existing data in it, nor to retrieve data from it. In particular, PRINT , SET with pointer-items, and WRITE are allowed, but not READ, REWRITE, DELETE, or INPUT.

If the access-mode is OUTIN, then all record operations (including REWRITE and DELETE) are allowed for the file.

The erase-statement shall be allowed only for a file with an access-mode of OUTIN.

If no access-mode is specified explicitly in the file-attribute-list, then the access-mode shall be OUTIN if the file can be both read and written, INPUT if it can only be read, and OUTPUT if it can only be written. Channel zero shall behave as if opened with OUTIN.

For a file opened with access-mode OUTPUT, the pointer shall be set to the end of file following the OPEN; otherwise, it shall be set to the beginning of file.

11.1.4.3 File-Organization. The file-organization specifies the logical relationship between file elements, and the means by which the file pointer can be manipulated to identify the elements. The organization is specified with one of the

keywords SEQUENTIAL, STREAM, RELATIVE, or KEYED, or with a string-expression whose value is one of these keywords. Devices are accessed as either SEQUENTIAL or STREAM. RELATIVE and KEYED are allowed only for true files.

A sequential file is a sequence of records. The order of the records is established by the order in which they were written. Records can be added only to the end of the file. The only means for identifying records with the file pointer is relative to the current position of the pointer, and the two special locations BEGIN (which identifies the first record in the sequence, if any), and END, immediately following the last record (the only location where it is possible to add records). A single record operation may affect several DISPLAY records, but only one INTERNAL or NATIVE record.

A stream file is much like a sequential file, except that it is a sequence of individual values, rather than of records. The order of values is established by the order in which they were written. Values can be added only to the end of the file. The only means for identifying values is relative to the current pointer position, or BEGIN and END (specifying, respectively, the first value, if any, in the sequence, and the location immediately following the last value). One record operation may typically read or write a contiguous series of values within a stream file.

A relative file is a sequence of record-areas, each of which may or may not contain a record. The record-areas are numbered sequentially beginning with 1. Thus the order of the record-areas and the records within them is established by the identifying integer associated with each. The file pointer may be manipulated with the use of this record number as well as by those means provided for sequential files. For relative files, the beginning of file is the first record-area, regardless of whether it contains a record. The end of file immediately follows the last existing record. Thus if the highest existing record number is 44, end of file refers to record-area 45. If there are no records in the file, end of file refers to record-area number 1. Records within a relative file may not only be read and written, but also changed (with REWRITE) and deleted (with DELETE). Moreover, records may be added, not only at the end of file, but also at any empty record-area, including those past the end of file. A record operation processes at most one record.

A keyed file is a sequence of records, each of which is identified by a string called a key. The logical sequence of

records is established by the collating order of their keys (cf. 11.1.4.6). The file pointer may be manipulated with respect to the keys, as well as by the means provided for sequential files. As with sequential files, beginning of file refers to the first existing record in the sequence (if any), and end of file refers to the location immediately following the last record. Records may be added anywhere within the sequence. An exact key, however, must always be specified for record creation, and no duplicate keys are allowed. Records may also be read, changed, or deleted. A record operation processes at most one record.

If no file-organization is explicitly specified in the open-statement, then the organization shall be determined from available system information about the file. If such information is insufficient, the system shall attempt to open the file as SEQUENTIAL. Channel zero shall behave as if opened with SEQUENTIAL.

11.1.4.4 Record-Types. A record-type specifies the logical representation of data within a record or as an individual file element. The record-type affects how data is interpreted and transformed when being transferred between a program and a file. A record-type is specified with one of the keywords DISPLAY, INTERNAL, or NATIVE, or with a string-expression whose value is one of these keywords.

The display type specifies that a record is a sequence of characters. On output, the characters are processed in accordance with the semantics of the PRINT statement, and on input with those of the INPUT statement (cf. Section 10). READ and WRITE are also allowed for display records; they follow the semantic rules for INPUT and PRINT, respectively.

The internal type specifies that a record is a sequence of typed values (or that each file element is a value), in the same sense that a program variable contains a value. The essential aspect of internal format is that (for a true file) values are preserved and retrievable. Thus, if a numeric or string value is written from a program variable, and later read into another variable, the two variables must be strictly equal (assuming the original variable to be unchanged). Since INPUT and PRINT statements are essentially character-oriented, they cause an exception when used on a file opened as internal.

The native type specifies that a record is a sequence of fields, as described by a program-specified TEMPLATE. This TEMPLATE, in conjunction with the list of operands of the associated record operation, specifies the size, type, number,

and order of fields within the record. This allows data in a file to be put in a form suitable for exchange with other language processors that have similar record specification capabilities. Values are preserved subject to certain restrictions regarding the size of the fields in the record. As with the internal type, INPUT and PRINT cause an exception when used on a file opened as native.

If no record-type is explicitly specified on the OPEN, the record-type is determined from available system information about the file. If such information is insufficient, then the file shall be opened as DISPLAY. Channel zero behaves as if opened with DISPLAY.

11.1.4.5 Record-Size. A record-size specifies the maximum length of records in a file. It is specified explicitly with the keyword LENGTH.

Unless an enhancement to this standard provides for fixed-length records, all files shall be composed of variable-length records (i.e., of records whose lengths are independent of each other). The length of a record of type DISPLAY shall be the number of characters in that record. The length of records of other types (INTERNAL and NATIVE) shall be implementation-defined. An attempt to perform a record operation for a record whose length exceeds the maximum set (either explicitly or by default) in the OPEN operation shall cause an exception. A specified LENGTH index shall be greater than zero.

If no record-size is explicitly specified in the open-statement, then the record-size is determined from available system information about the file. If such information is insufficient, then the file shall be opened as VARIABLE. If the index is omitted, then the maximum length of records shall be implementation-defined. Channel zero shall behave as if opened with VARIABLE and the length index omitted.

Implementations must support record-sizes of at least 132 for true files.

11.1.4.6 Collate-Sequence. The collate-sequence specifies, for a KEYED file, the collating sequence of the record keys. A collate-sequence is specified with one of the keywords STANDARD or NATIVE, or with a string-expression whose value is one of these keywords. Collate-sequence has meaning only for a KEYED file. For other file-organizations, it has no effect.

The collate-sequence of a file governs all record operations and the file operation ERASE for that file. Thus, the logical appearance of the file, when operated on by READ, WRITE, REWRITE, DELETE, SET with pointer-control, ERASE, and ASK shall be in accordance with the specified collate-sequence (cf. 11.1.4.3 and 11.2). The collate-sequences STANDARD and NATIVE imply exactly the same ordering as in the option-statement (cf. 6.6). Thus, if the collate-sequence associated with a file and a program-unit agree, it follows that an earlier key in the file will always compare as less than a later key. When the sequences disagree, this relationship may not hold. Nonetheless, it shall be possible for a program-unit with a different collate-sequence to access a KEYED file; the collate-sequence affects only the logical order of the records, not their contents. Implementations with KEYED files must support both collate-sequences.

If no collate-sequence is specified in the open-statement, then the collate-sequence shall be determined from available system information about the file. If such information is insufficient, the system shall attempt to open the file with the same collate-sequence as that in effect for the program-unit containing the open-statement. Since channel zero has file-organization SEQUENTIAL (not KEYED), it has no associated collate-sequence.

11.1.4.7 Close-Statement. Execution of a close-statement shall close the file assigned to the specified channel, causing the channel to become inactive. If no file is assigned to the channel, no action occurs. Upon exit from an external-sub-def, external-function-def, or external-picture-def, or on termination of a parallel-section, any files opened by such a procedure whose channels are not formal parameters shall be closed. Upon program termination, any files still open shall be closed.

11.1.4.8 Erase-Statement. For a true file, execution of an erase-statement shall delete all or part of the data within the file assigned to the specified channel. The file-attributes associated with the file are not changed. If the REST option is omitted, then all file elements are deleted (or records deleted from record-areas, for RELATIVE files), the file becomes empty, and the file pointer points to the end of file (which is the same as the beginning of file).

If the REST option is specified, then all file elements at or beyond the current location of the file pointer are deleted. All file elements preceding it are left unchanged. The file pointer is then set to end of file.

The erase-statement may not be effective for a device. The ask-statement can be used to determine if a device supports this capability.

An erase-statement executed for channel zero shall cause an exception, but no other effect shall occur.

An erase-statement is allowed only for a file opened with access-mode OUTIN. For other access-modes, there is no effect on the file and an exception results.

11.1.4.9 Ask-Statement. Execution of an ask-statement shall cause the variables in the ask-item-list to be assigned values corresponding to the attributes of the file currently assigned to the specified channel, as indicated in Table 5. If the channel is inactive, then all such string-variables shall be assigned the null string, and all such numeric variables shall be assigned 0. In Table 5, A$ represents a string-variable and N represents a numeric-variable.

Table 5. Values for Ask-Statement

Ask-item	Values
ACCESS A$	The access-mode of the file (i.e., "INPUT", "OUTPUT", or "OUTIN").
COLLATE A$	The collate-sequence associated with a KEYED file (i.e., "standard" or "NATIVE"). For file-organizations other than KEYED, the null string is assigned.
DATUM A$	The type of the next datum in the file following the current pointer position (i.e., "NUMERIC", "STRING", "NONE" (if no data follow), or "UNKNOWN" (if it is impossible to determine the type or whether more data follow)). DATUM is well-defined only for STREAM INTERNAL files. For other file organizations, it is implementation-defined.
ERASABLE A$	"YES" or "NO" depending on whether or not this file is erasable, i.e., if the ERASE statement can delete file elements.

FILETYPE A$	"FILE" or "DEVICE" depending on whether this is a true file capable of preserving data, or is a device.
KEY A$	The key associated with the record identified by the file pointer in a keyed file. If the pointer is at the end of file or if this is is not a keyed file, the null string is assigned.
MARGIN N	The current margin for a display file (MAXNUM if the record may be of arbitrary length). If the file is not DISPLAY, zero is assigned.
NAME A$	The name of the file assigned to the channel.
ORGANIZATION A$	The file-organization of the file (i.e., "SEQUENTIAL," "STREAM," "RELATIVE," or "KEYED").
POINTER A$	The current pointer position for the file (i.e., "BEGIN," "MIDDLE," or "END," where MIDDLE shall mean neither BEGIN nor END, and END shall be the pointer position for an empty file, or a position beyond the end, in the case of a RELATIVE file). UNKNOWN may be returned in the case of devices for which an implementation cannot determine which of the above values is correct.
RECORD N	The number of the record-area identified by the file pointer. For non-RELATIVE files, zero is assigned.
RECSIZE A$ N	The record-size of the file (e.g., "VARIABLE") and the maximum length for its records (MAXNUM if there is no effective limit on record-length, as with, e.g., a communication line).
RECTYPE A$	The record-type for the file (i.e., "DISPLAY," "INTERNAL," or "NATIVE").
SETTER A$	"YES" or "NO" depending on whether or not this file has record-setter capability.
ZONEWIDTH N	For DISPLAY files, the current zonewidth. For non-DISPLAY files, zero is returned.

Table 6 shows the effect of executing an ask-statement for channel zero.

Table 6. Ask-Statement with Channel Zero

Ask-Attribute	Value
ACCESS	OUTIN
COLLATE	null
DATUM	UNKNOWN
ERASABLE	NO
FILETYPE	DEVICE
KEY	null
MARGIN	current margin
NAME	implementation-defined
ORGANIZATION	SEQUENTIAL
POINTER	UNKNOWN
RECORD	0
RECSIZE	VARIABLE MAXNUM
RECTYPE	DISPLAY
SETTER	NO
ZONEWIDTH	current zonewidth

11.1.5 Exceptions

The value of a channel-expression is not between 0 and the implementation-defined maximum (7001, fatal).

Channel zero is specified in an open-statement, a close-statement, or an erase-statement (7002, nonfatal: do nothing and continue).

A nonzero channel specified in an open-statement is already active (7003, fatal).

A string-expression used to specify a file-attribute does not have a recognizable value (7100, fatal).

Access to a file in an open-statement is not possible in accordance with the specified or default file-attributes (71xx, fatal: the values and meanings for xx are implementation-defined).

A KEYED file is re-opened with a different collate-sequence from that of an earlier open (7050, fatal).

A LENGTH index is not greater than zero (7051, fatal).

A device is opened as RELATIVE or KEYED (7052, fatal).

A nonzero channel specified in an erase-statement is inactive (7004, fatal).

An erase-statement is used on a file that has not been opened as OUTIN (7301, fatal).

An erase-statement is used on a device without erase capability that has been opened with OUTIN (7311, nonfatal: do nothing and continue).

11.1.6 Remarks

It is recommended that implementations recognize as file-names at least those strings of characters consisting of an upper-case-letter followed by at most three more upper-case-letters or digits. It is also recommended that information required by the operating system for the purpose of protecting the security of files be considered part of the file-name.

It is recommended that implementations use the file-name to distinguish between the opening of a true file, and opening of non-file devices, such as a communications line or a line printer.

It is recommended that the number of channels that may be active simultaneously be at least four in addition to channel zero.

It is recommended that the default maximum length of records in a file be infinite (i.e., that records be allowed to be of any length).

It is also recommended that record-size for INTERNAL and NATIVE have a meaning comparable to that for DISPLAY (i.e., that it specifies the maximum number of characters or bytes within the record).

Additional values may be returned by an ASK statement if an implementation supports access-modes, file-organizations, record-types, record-sizes, and collate-sequences in addition to those specified in this standard.

If implementations return a status code following various file operations, it is recommended that this be made accessible

through an additional ASK attribute to be called IOSTAT which returns a single string value, e.g., "ASK IOSTAT S$" returns a value in S$ reflecting the status of the file following the last attempted operation.

The maximum length of a key is implementation-defined.

11.2 File Pointer Manipulation

11.2.1 General Description

The pointer for an open file can be altered in certain ways, without also performing any data transfer. The rules for pointer manipulation with the set-statement with pointer-items also apply when used in conjunction with other record operations.

11.2.2 Syntax

Core productions:

```
 1. set-object           > channel-setter pointer-items
 2. pointer-items        = (pointer-control / io-recovery /
                           pointer-control comma io-recovery)
 3. pointer-control      > POINTER core-record-setter
 4. record-setter        > core-record-setter
 5. core-record-setter   = BEGIN / END / NEXT / SAME
 6. io-recovery          = missing-recovery /
                           not-missing-recovery
 7. not-missing-recovery = IF THERE THEN io-recovery-action
```

Enhanced Files productions:

```
 8. pointer-control      > enhanced-record-setter
 9. record-setter        > enhanced-record-setter
10. enhanced-record-setter = RECORD index /
                           KEY (exact-search / inexact-search)
                           string-expression
11. exact-search         = equals-sign?
12. inexact-search       = greater-than-sign / not-less
```

11.2.3 Examples

```
1. SET #N: POINTER BEGIN
   SET #3: RECORD N+1, IF MISSING THEN 1200
   SET #4: KEY "Jones", IF THERE THEN EXIT DO
```

11.2.4 Semantics

Execution of a set-statement with pointer-items shall set the pointer for the file assigned to the specified channel. After the pointer has been set, an optional io-recovery may take effect. The semantics associated with the record-setter and when the io-recovery takes effect are uniform for all the record operations (cf. 11.3, 11.4, and 11.5). If any of the exceptions listed in 11.2.5 occurs, the file pointer remains unchanged from

its state before the SET with pointer-control. A device may not be able to achieve the effect of a record-setter. The ask-statement may be used to determine whether a device has record-setter capability.

11.2.4.1 Record-Setters. An absent record-setter leaves the file pointer unchanged from its previous state. The io-recovery (see 11.2.4.2), if present, still has its usual effect.

A record-setter of NEXT indicates that the pointer is to be set to the next existing record (for non-STREAM files) or value (for STREAM files) at or beyond the current location. In the case of a RELATIVE file, NEXT shall therefore cause the pointer to skip over any empty record areas to the next existing record. If the pointer is already at or beyond the end of the file, or is pointing to an existing record, NEXT shall leave the pointer unchanged. This capability allows RELATIVE files to be processed as if they were SEQUENTIAL. In the case of STREAM and KEYED files, the pointer is always pointing to an existing file element or end of file and so is left unchanged. For a SEQUENTIAL file, the only case in which NEXT would have some effect is if there were a partial record pending (cf. 11.3). In this case, an end-of-record shall be generated and the pointer left at end of file.

A record-setter of BEGIN causes the pointer to be set to the beginning of file (i.e., to the first file element). If the file is empty, this location is also the end of file.

A record-setter of END causes the pointer to be set to end of file, defined as immediately beyond the last file element (if any) in the case of SEQUENTIAL, STREAM, and KEYED files, and as immediately beyond the last existing record in the case of a RELATIVE file (or at record-area number 1, if no records exist).

A record-setter of SAME allows the user to access the same file-element(s) that have most recently been processed since the OPEN for that channel. Its use is valid only if the most recently executed record operation that accessed the channel meets these conditions: (1) it was not a delete-statement, and (2) no exception occurred during its execution at least until after the file pointer had been set. If these conditions are not met, no pointer manipulation takes place and an exception results. If they are met and the most recent operation was a READ, INPUT, SET with pointer-items, or REWRITE, then the file pointer is reset to the same file element it was just set to by the record-setter of that operation. If this operation had no record-setter, then SAME resets the pointer to the same location

it had at the beginning of that operation. If the most recent operation was a WRITE or PRINT, then SAME sets the pointer to the first file element created by that operation.

A record-setter with RECORD is valid only for use with RELATIVE files. If an attempt is made to use this record-setter on a file not opened as RELATIVE, the pointer is left unchanged and an exception results. The index is evaluated by rounding to an integer, and the pointer set to the corresponding record-area, whether or not it contains a record. If the index evaluates to an integer less than one, the pointer is left unchanged and an exception is generated.

A record-setter with KEY is valid only for use with a KEYED file. If an attempt is made to use this record-setter on a file not opened as KEYED, the pointer is left unchanged and an exception results. For an exact-search the pointer is set to the record whose key equals that of the string-expression; if none such exists, the pointer is set to the first record whose key is greater than the string-expression. If there is no such record, the pointer is set to end of file. For an inexact-search with not-less, the pointer is set exactly as for an exact-search, except for the setting of the data-found condition (see 11.2.4.2). For an inexact-search with greater-than-sign, the pointer is set to the first record whose key is strictly greater than the string-expression; if none exists, it is set to end of file.

11.2.4.2 Io-Recovery. At the completion of pointer manipulation there shall be set a condition called data-found, which is either true or false. If data-found is true, and if a not-missing-recovery has been specified, then the io-recovery-action takes effect. If the data-found condition is false and a missing-recovery has been specified, then the io-recovery-action also takes effect. Except for these two cases, the io-recovery-action, if any, is ignored. The data-found condition is false if:

(1) An exact-search has been specified, but no record was found whose key was equal to the string-expression
(2) After the pointer is set, it points to end of file
(3) After the pointer is set, it points to an empty record-area
(4) For a device, there is an implementation-defined condition signifying that no data is available for input

Otherwise, the data-found condition is true.

If the io-recovery-action is an exit-do- or exit-for-statement, the statement shall have its normal effect (cf. 8.3). If the io-recovery-action is a line-number, then execution shall continue at the specified line.

11.2.5 Exceptions

A set-statement with pointer-items is executed for an inactive channel (7004, fatal).

A record-setter is used with channel zero (7002, nonfatal: do nothing and continue).

A record-setter is used on a device without record-setter capability (7205, nonfatal: do nothing and continue).

The record-setter SAME is used, and the most recently executed operation for the channel was a delete-statement (7204, fatal).

The record-setter SAME is used, and the most recently executed operation for the channel caused an exception before pointer manipulation took place (7204, fatal).

The record-setter SAME is used, and no record operation has been executed on that channel since the OPEN (7204, fatal).

The record-setter RECORD is used on a file opened with a file-organization other than RELATIVE (7202, fatal).

The record-setter KEY is used on a file opened with a file-organization other than KEYED (7203, fatal).

The index of a RECORD record-setter evaluates to an integer less than one (7206, fatal).

A record-setter specifies an exact-search for the null string (7207, fatal).

11.2.6 Remarks

For devices, data-found could be set false by such conditions as no more cards in a card reader, control-Z sent from a terminal (which signifies end of file for some systems), or that the device is for output only (e.g., a line printer).

Given a RELATIVE file, with three existing records, numbers 2, 5, and 6:

```
 1        2        3        4        5        6        7
---     RECORD     ---      ---     RECORD   RECORD    ---
                    ^
                    |
                 pointer
```

if READ RECORD 2 has just taken place, then the following record-setters will have the described effect:

record-setter	resulting pointer position in front of record number
BEGIN	1
END	7
SAME	2
NEXT	5
absent	3
RECORD n	n

11.3 File Data Creation

11.3.1 General Description

Statements are provided to allow the user to send data developed within the program to an external destination. In the case of true files, such data can be retrieved and modified by later programs. The facilities generalize the output capabilities presented in Section 10 to files. New facilities are also defined to allow output to the various record-types. The set-objects MARGIN or ZONEWIDTH are not part of a data creation statement, but are included in this section because of their interaction with display records.

11.3.2 Syntax

Core productions:

```
 1. print-statement          > PRINT channel-expression
                               print-control
                               (colon (print-list / output-list))?
 2. array-print-statement    > MAT PRINT channel-expression
                               print-control colon
                               (array-print-list /
                               array-output-list)
 3. print-control            = (comma print-control-item)*
 4. print-control-item       = core-record-setter /
                               not-missing-recovery / USING image
 5. set-object               > channel-setter (MARGIN / ZONEWIDTH)
                               index
 6. write-statement          = WRITE channel-expression
                               write-control colon expression-list
 7. array-write-statement    = MAT WRITE channel-expression
                               write-control colon array-list
 8. write-control            = (comma write-control-item)*
 9. write-control-item       > record-setter /
                               not-missing-recovery
10. expression-list          = expression (comma expression)*
11. array-list               = array-name (comma array-name)*
```

Enhanced Files productions:

```
N12. write-control-item      > template-identifier
N13. template-identifier     = WITH (line-number /
                               string-expression)
N14. declarative-statement   > template-statement
N15. template-statement      = TEMPLATE colon
                               template-element-list
```

```
N16. template-element-list  = template-element
                              (comma template-element)*
N17. template-element       = fixed-field-count
                              (field-specifier /
                              left-parenthesis
                              template-element-list
                              right-parenthesis) /
                              variable-field-count
                              field-specifier
N18. fixed-field-count      = SKIP? (integer OF)?
N19. variable-field-count   = question-mark OF
N20. field-specifier        = numeric-specifier /
                              string-specifier
N21. numeric-specifier      = NUMERIC asterisk
                              numeric-field-size
N22. numeric-field-size     = fixed-point-size / E
N23. fixed-point-size       = integer-size period? /
                              integer-size? period fraction-size
N24. integer-size           = integer
N25. fraction-size          = integer
N26. string-specifier       = STRING asterisk string-field-size
N27. string-field-size      = integer
```

Within a print-statement or array-print-statement, an image shall not be used with a print-list (i.e., only an output-list shall be used when an image is present as a print-control).

The line-number of a template-identifier shall refer to a template-statement in the same program-unit.

The integer in a fixed-field-count shall be greater than zero.

In a fixed-point-size, the integer-size or fraction-size shall be greater than zero.

String-field-size shall be greater than zero.

The record-setter in a write-control shall not specify an inexact-search.

A given print-control-item shall appear at most once in print-control.

A given write-control-item shall appear at most once in write-control.

11.3.3 Examples

```
 1. PRINT #3: A,B,C
    PRINT #3, END, USING 123: A$, B+C;
 2. MAT PRINT #N, SAME, IF THERE THEN EXIT FOR: A$; B$, C
 5. #3: MARGIN N+1
 6. WRITE #3, RECORD 47, IF THERE THEN 666: A+B, C$ & D$
    WRITE #X+Y WITH TEMPLATE3$: X, Y, Z + W
 7. MAT WRITE #3, KEY "Whoever", IF THERE THEN 666,
                                  WITH 111: A, B$
N15. TEMPLATE: STRING*5, 2 OF NUMERIC*3.4
     TEMPLATE: ? OF NUMERIC*5.2, ? OF STRING*5
N16. 5 OF STRING*22, 3 OF NUMERIC*E, ? OF NUMERIC*.6
```

11.3.4 Semantics

All data creation statements follow a general pattern. In all cases, the function of a data creation statement is to add one or more new file elements to a file. Previously existing file elements are not affected. Details on aspects peculiar to each of the various forms are presented in 11.3.4.1 -- 11.3.4.12 under the headings of each statement type.

First, the channel to which the data will be sent is determined from the channel-expression. The file-attributes are checked against the intended operation. All data creation statements require an access-mode of OUTPUT or OUTIN. If the channel is active and the file-attributes are compatible with the data creation statement, then the next phase begins. Otherwise, an exception results and the file, the file pointer, and all program variables remain unchanged.

The second phase of processing involves setting the file pointer, based on the record-setter (or its absence). This is done exactly as described in 11.2.4.1. The data-found condition is now set, as described in 11.2.4.2. If data-found is true and a not-missing-recovery is present, then the io-recovery-action is taken. If data-found is true and a not-missing-recovery is absent, then an exception results. In either case, no further change is made to the pointer position or the file.

If data-found is false, then the third phase begins, the actual output of data at the location indicated by the pointer. The operands are evaluated in succession from left to right until enough data to fill a file-element has been generated. Only then is the file-element actually added to the file and the pointer advanced immediately beyond the file-element just created. In particular, this means that an exception during data transfer

will never result in a partial file-element being added to the file. However, if a statement can create several file-elements, those that have already been created before the exception occurs continue to exist in the file. Following the completion of data transfer, the file pointer is always left pointing at the next file-element (or end of file if none such exists) beyond the last one created. If an exception prevented the creation of any file elements, the pointer is left as it was set in the second phase.

11.3.4.1 Print-Statement. The transfer of data with the print-statement works just as described in 10.3 and 10.4, except that the sequence of characters generated constitutes a record of a DISPLAY file, rather than current line, and end-of-record is generated in place of end-of-line. Note that it is possible to create records containing zero characters. End of record is the implementation-defined means whereby it is indicated that the storage of a file-element in a file is completed (i.e., no change or addition to this record is possible with a data creation statement). Except for the special case discussed in the next paragraph, no data is actually added to the file until a valid end-of-record has been generated (i.e., partial records are not added to the file). The effective margin for the file, which is used to control when end-of-record is generated, is taken from the value of RECSIZE established when the file was opened, or from a set-statement with MARGIN for this channel executed since the open (cf. 11.3.4.3).

In one special case, a partial record is added to a file. If the print-list or output-list contains a trailing print-separator, then upon successful completion of the statement, a partial record has been created at the end of file, with the pointer left at end of file (i.e., just beyond the partial record). Note that if there is an exception before completion of the statement, then the partial record is not added to the file, as would also be true of a complete record. If and only if the next operation for the channel is a print-statement with an absent record-setter, then the sequence of characters it generates is appended to the end of the partial record, in accordance with the usual rules regarding the margin. If the next operation for that channel is anything other than such a print-statement, then before any other processing takes place, an end-of-record is added to complete the partial record, and the pointer left at end of file.

11.3.4.2 Array-Print-Statement. The transfer of data with the array-print-statement works just as described in 10.5, except that the sequence of characters generated constitutes a record of a DISPLAY file, rather than the current line, and end-of-record

is generated in place of end-of-line. Note that by the rules of 10.5, a partial record is never created by the array-print-statement with an array-print-list, but may be created with an array-output-list containing a trailing semicolon. Since an array-print-statement always starts a new line, it may not be used to complete a partial record. The effective margin shall be controlled as described in 11.3.4.1 and 11.3.4.3.

11.3.4.3 Setting the Margin. Associated with each open DISPLAY file is a margin, the maximum number of characters that a record can contain. The margin is used by the print- and array-print-statement to determine when an end-of-record should be generated. Upon open, the effective margin is the LENGTH index in RECSIZE, or the implementation-defined length, which shall be not less than the default zone width, if LENGTH is not specified explicitly. The set-statement with a MARGIN changes the margin for the active channel to the specified index. If a partial record exists in the file affected by the margin, the new margin is used when subsequently attempting to complete the partial record. A margin of MAXNUM indicates that a record may be arbitrarily large. The margin index shall evaluate to greater than zero. The effect of a set-statement with MARGIN ends when the file is closed.

The maximum margin supported is implementation-defined.

11.3.4.4 Setting the Zone Width. Associated with each open DISPLAY file is a zone width, which controls the effect of PRINT as described in 10.3. Upon open, the zone width for a file is set to the implementation-defined default value, which shall be at least d+e+6. The set-statement with ZONEWIDTH changes the zone width for the active channel to the specified index.

All zones are the same width, except possibly the last, which may be shorter. If a partial record exists in the file affected by the set-statement with ZONEWIDTH, the new zone width is used when subsequently attempting to complete the partial record. In a set-statement with ZONEWIDTH, the index shall evaluate to greater than zero. The effect of the set-statement with ZONEWIDTH ends when the file is closed.

The maximum zone width is implementation-defined.

11.3.4.5 Display Record-Type. DISPLAY records are sequences of characters; the characters generated by the PRINT operations are as described in Section 10. The accuracy of numeric values is limited only by the implementation-defined

significance-width or length of the format-item, and by the ARITHMETIC option in effect.

11.3.4.6 Channel Zero. PRINT to channel zero works in accordance with the semantics for a device without record-setter or erase capability. The destination of the output data is the same as if the channel-expression had been omitted. Also, if there is an exception for which a different recovery procedure is specified in Section 10 than in Section 11, the procedure of Section 10 shall be used. A set-statement with MARGIN or ZONEWIDTH specifying channel zero has the same effect as if the channel-setter were omitted.

11.3.4.7 Write Operation. The write- and array-write-statements are used to create records of any type. Successive expression values are arranged in corresponding sequences of values or fields or characters, and written out to the file. Partial records are never created.

11.3.4.8 File Organization. For STREAM files, one statement may create several file-elements, specifically one file-element is created for each expression or array-element evaluated. If an exception occurs during internal evaluation, previously created values remain in the file, the pointer is left at end of file, and any remaining values are ignored.

For non-STREAM, non-DISPLAY, files, the values or fields generated by the expressions or arrays form one record. Thus, if an exception occurs before the statement is completed, no record is added to the file, and the file pointer is left unchanged from phase two.

Since records in a KEYED file are identified by their keys, it is necessary when creating a new record that an explicit key be associated with it; thus when attempting to write to a KEYED file, an exact-search shall always be specified in the record-setter. If data-found is false, then a new record is inserted into the file, with a key equal to the string-expression of the exact-search.

11.3.4.9 Display Record-Type. A WRITE operation on a display record generates exactly the same sequence of records and characters as would a PRINT operation with the same expression-list or array-list. Note, however, that not all PRINT facilities are available for WRITE, which is record-oriented rather than line-oriented.

For DISPLAY files, one statement may create several file-elements in the case of margin overflow. If a fatal exception occurs during generation of an element, any elements previously created by that statement remain in the file, the pointer is left at the end of the file, and any remaining expressions are ignored.

11.3.4.10 Internal Record-Type. An internal record (and a STREAM file) is a sequence of values. There are two types of values, numeric and string. The sequence of values and their types are determined by the sequence and types of the expressions and array-elements from which the values are generated. When a value stored in a file is later retrieved, the effect is as if the expression or array-element with which it was created were assigned to the input variable with a let-statement. The length and content of string values are preserved. Numeric values are also preserved consistent with the usual limitations on precision associated with the prevailing ARITHMETIC option.

11.3.4.11 Native Record-Type. The TEMPLATE describes the location, size, and type of fields within a record. When writing to a native file, a TEMPLATE shall always be used. It shall not be used with any other record-type. A TEMPLATE shall be associated with a particular data creation statement by means of the template-identifier. The template-identifier specifies either the template-statement to be used or a string-expression whose value shall be a syntactically correct template-element-list. The string-expression is evaluated before the expression-list or array-list of the write- or array-write-statement. Several statements may use the same template-statement.

When generating data for a native record, each expression within the expression-list is associated with a numeric- or string-specifier within a TEMPLATE; the specifier is then used to transform the value of the expression into a field within a record. The association takes place from left to right within the expression-list and the template-element-list, each expression using the next available field-specifier. If the type of the expression (numeric or string) disagrees with the type of the specifier, an exception results. The number of expressions shall not be greater than the number of specifiers. Extra specifiers beyond the last expression are ignored. The contents of the field are determined by the value of the expression and the size characteristics of the specifier.

For string values, the string-field-size is the number of characters in the field. The string value is left-justified within the field. If the value's length exceeds that of the

field, an exception results. If shorter, the field is padded on the right with spaces.

Numeric fields are used to retain numeric values (both magnitude and sign). The sign is always stored in the field, but the numeric-field-size explicitly describes only the storage of the number's magnitude. For a numeric-field-size of E, the value is retained with the implementation-defined number of significant digits for the prevailing ARITHMETIC option. For a fixed-point-size, the integer-size describes the number of available digit places to the left of the decimal point, and the fraction-size, to the right of the decimal point. An omitted integer-size or fraction-size is treated as equivalent to zero. The numeric value is stored in accordance with these sizes. If the value contains significant digits to the right of the available field positions, the value is rounded when stored. This may result in a field with a value of zero. If the value contains significant digits to the left of the available field positions, an exception results.

The fixed-field-count in a template-statement indicates skipping and/or repetition for individual specifiers or a series of specifiers. The integer of the fixed-field-count indicates the number of repetitions and the keyword SKIP indicates that the affected specifiers shall generate skipped fields. A field-count applies to the entity (either a field-specifier or a parenthesized template-element-list) in the same template-element. For an integer field-count (indicating repetition) the effect is just as if the entity governed by the field-count had been written out explicitly the equivalent number of times. When SKIP is used within a field-count, it indicates that the specifiers governed by it are not associated with values from the expression-list or array-list. Rather, as the record is being generated, fields within the record are assigned the value zero if numeric and spaces if string, corresponding to the usual size and type of the field-specifier in question. If a field-specifier is governed by several SKIPs (from various levels above it) the effect is just as if it were governed by only one.

For example, given the following expression-lists and templates-element-lists,

```
A,B,C   and   NUMERIC*4, NUMERIC*5, NUMERIC*6
D,E     and   NUMERIC*4, SKIP NUMERIC*5, NUMERIC*6
```

A and D will occupy equivalent field locations, as will C and E. The second field in the second record will be the same size as occupied by B in the first record, with a value of zero.

As further illustrations of the preceding description, equivalent pairs of template-element-lists are shown below:

```
(1) STRING*4, STRING*4, STRING*4
(2) 3 OF STRING*4

(1) STRING*5, STRING*4, NUMERIC*2, NUMERIC*2, NUMERIC*2,
      STRING*4, NUMERIC*2, NUMERIC*2, NUMERIC*2
(2) STRING*5, 2 OF (STRING*4, 3 OF NUMERIC*2)

(1) SKIP NUMERIC*3, SKIP NUMERIC*3, NUMERIC*4, SKIP STRING*5,
      SKIP STRING*6
(2) SKIP 2 OF NUMERIC*3, NUMERIC*4, SKIP (STRING*5,
      SKIP STRING*6)
```

Note that in (2) of the last example, the SKIP immediately in front of STRING*6 is superfluous. A variable-field-count is used only in conjunction with arrays (see 11.3.4.12).

If execution reaches a template-statement, it proceeds to the next line with no further effect.

11.3.4.12 Array-Write-Statement. An array-write-statement for the INTERNAL and NATIVE record-types behaves just like the write-statement would if the arrays were written out explicitly as array-elements in row major order (the last subscript varying most rapidly). Thus, for example, if "DIM A(3), B(2,2)" and OPTION BASE 1 are in effect, the following two statements are equivalent:

```
MAT WRITE #3: A, B
WRITE #3: A(1),A(2),A(3), B(1,1),B(1,2),B(2,1),B(2,2)
```

When writing to a NATIVE record, arrays in an array-list can use the variable-field-count. If a fixed-field-count is used, then the number and type of the specifiers shall match the array-elements as with WRITE. When the first element of an array is to be associated with a field-specifier, and if a template-element has just been completed (or if this is the first array in the list), and if the next template-element has a variable-field-count, then the field-specifier is used for all the elements of the array. When evaluation of the array is complete, the next array, if any, uses the next template-element, which may or may not also have a variable-field-count. An array shall use either a template-element with a variable-field-count or template-elements with fixed-field-counts, but not both.

11.3.5 Exceptions

The value of the index in a set-statement with MARGIN is less than the current zonewidth for that file (4006, fatal).

The value of the index in a set-statement with a ZONEWIDTH is less than one, or greater than the current margin for that file (4007, fatal).

A set-statement with a MARGIN or ZONEWIDTH specifies an inactive channel (7004, fatal).

A set-statement with a MARGIN or ZONEWIDTH specifies a file not opened as DISPLAY (7312, fatal).

A set-statement with a MARGIN or ZONEWIDTH specifies a file opened as INPUT (7313, fatal).

The following exceptions for data creation statements are grouped according to the phase of processing during which they are detected. Phase 1 exceptions imply no change to the file or file pointer. Phase 2 exceptions imply no change to the file. Phase 3 exceptions imply that some file-elements may have been created.

11.3.5.1 Phase 1 Exceptions

A data creation statement attempts to access an inactive channel (7004, fatal).

A print- or array-print-statement attempts to access a file opened as INTERNAL or NATIVE (7317, fatal).

The record-setter cannot be processed correctly, as described in 11.2.5 (7002 and 7202-7207, use the procedures of 11.2.5).

A data creation statement attempts to access a file opened as INPUT (7302, fatal).

A write- or array-write-statement attempts to access a KEYED file, but does not specify an exact-search in its record-setter (7314, fatal).

The string-expression of a template-identifier is not a syntactically correct template-element-list (8251, fatal).

A template-identifier is used on a file opened as DISPLAY or INTERNAL (7315, fatal).

A write- or array-write-statement does not have a template-identifier when attempting to access a file opened as NATIVE (7316, fatal).

11.3.5.2 Phase 2 Exception

For a data creation statement, the condition data-found is true, and a not-missing-recovery has not been specified (7308, fatal).

11.3.5.3 Phase 3 Exceptions

An attempt is made to create a record larger than the value of RECSIZE (8301, fatal).

An expression or array-element does not agree in type (numeric or string) with its associated TEMPLATE field-specifier (8252, fatal).

A template-element with a variable-field-count does not coincide with the first element of an array (8253, fatal).

There are not enough field-specifiers in a template-statement for all the expressions or array-elements (8254, fatal).

A numeric value has significant digits to the left of the available digit places in the field of a template (8255, fatal).

A string value is longer than the length of its field in the template (8256, fatal).

11.3.6 Remarks

Implementations may provide syntactic enhancements to template-element-list, e.g., to allow for additional data types. The exception for incorrect syntax then applies to the enhanced definition of template-element-list.

The variable-field-count is especially useful when writing an array whose size may change in the program, since the use of the fixed-field-count implies knowing the exact size in advance.

11.4 File Data Retrieval

11.4.1 General Description

Statements are provided to allow the program to retrieve data from a file to which it has previously been written or from a device. The facilities generalize the input capabilities presented in Section 10 to files. New facilities are also defined to allow input from the various record-types.

11.4.2 Syntax

Core productions:

```
 1. input-statement          > INPUT channel-expression
                               input-control colon variable-list
                               (comma SKIP REST)?
 2. array-input-statement    > MAT INPUT channel-expression
                               input-control colon
                               (redim-array-list /
                               variable-length-vector)
 3. line-input-statement     > LINE INPUT channel-expression
                               input-control colon
                               string-variable-list
 4. array-line-input-statement > MAT LINE INPUT
                               channel-expression input-control
                               colon redim-string-array-list
 5. input-control            = (comma input-control-item)*
 6. input-control-item       = core-record-setter /
                               missing-recovery /
                               prompt-specifier /
                               timeout-expression / time-inquiry
 7. read-statement           > READ channel-expression
                               read-control colon variable-list
                               (comma SKIP REST)?
 8. array-read-statement     > MAT READ channel-expression
                               read-control colon
                               redim-array-list
 9. read-control             = (comma read-control-item)*
10. read-control-item        > record-setter / missing-recovery
```

Enhanced Files productions:

```
N11. read-control-item       > template-identifier
```

The line-number of a template-identifier shall refer to a template-statement in the same program-unit.

A given input-control-item shall appear at most once in input-control.

A given read-control-item shall appear at most once in read-control.

A variable-length-vector shall be declared as one-dimensional.

11.4.3 Examples

```
1. INPUT #3: A,B,C,A$
2. MAT INPUT #W, BEGIN, IF MISSING THEN EXIT DO: A,B$
3. LINE INPUT #Q, NEXT: A$, B$, C$
4. MAT LINE INPUT #4, IF MISSING THEN 1234: A$, B$(N), C$(8)
7. READ #3, SAME, WITH 333: W$, SKIP REST
8. MAT READ #N, RECORD W+2, IF MISSING THEN 111,
                            WITH 222: N, W(Q)
```

11.4.4 Semantics

All data retrieval statements follow a general pattern. Details on the aspects peculiar to each of the various forms are presented in 11.4.4.1-11.4.4.12 under the headings for each statement type.

First, the channel from which data will be retrieved is determined from the channel-expression. Then, the file-attributes are checked against the intended operation. All data retrieval statements require an access-mode of INPUT or OUTIN. If the channel is active and the file-attributes are compatible with the data retrieval statement, then the next phase begins. Otherwise, an exception results and the pointer and all program variables remain unchanged.

The second phase of processing involves setting the file pointer, based on the record-setter if present. In the absence of a record-setter the file pointer does not change. This is done exactly as described in 11.2. The data-found condition is now set, again as described in 11.2. If data-found is false and a missing-recovery is present, then the io-recovery-action is taken, otherwise an exception results. In either case, no further change is made to the pointer position.

If data-found is true, then the third phase begins, the actual input of data from the element indicated by the pointer. Data is transferred from the file element(s) to each of the operands (variables or arrays), from left to right, with

successive data or values or fields from the file being assigned to successive variables or arrays. Note in particular that evaluation of subscripts, substring-qualifiers, and redims is delayed until after assignment of data to previous operands, but occurs before assignment of data to the operand to which they apply. Note also that assignment of a string value to a string variable with a substring-qualifier takes place in accordance with the usual semantics of string assignment described in 6.5. If an exception occurs during data transfer, variables and array-elements for which a legal assignment has been made retain their new values, but all subsequent variables and array-elements retain their original values. Following a successful data retrieval operation, the pointer is advanced to the next file element, that is, the next record, record-area, or value in the file.

One data retrieval operation usually affects only one file element. The three cases in which several file elements may be processed are: (1) LINE or MAT LINE INPUT, (2) READing from a STREAM file, and (3) INPUT or READ from a DISPLAY file with records with trailing commas (indicating continuation of data).

The SKIP REST option is allowed only for non-STREAM files. It causes the remainder of the record from which the last datum or value or field was taken to be ignored. If a TEMPLATE is being used, it also causes any remaining specifiers to be ignored. It is still mandatory that the record contain enough data to satisfy the variables or arrays in the list.

During this third phase of processing, a number of exception conditions may arise. Each such exception is associated with a particular file element. In all cases, the pointer is advanced to the file element immediately following the one with which the exception is associated. Table 7 summarizes these exceptions and the file-element to which they apply.

Table 7. Association of File-Element with Exception

Exception	Associated file-element
File-element larger than RECSIZE	The oversize file-element

Invalid redim, subscript, substring-qualifier, redim too large.	The file-element from which data would have been taken
Bad TEMPLATE: wrong type, field-count of "?" on other than first element of an array, too few specifiers	The file-element from which data would have been taken
Bad data: wrong type, syntax, overflow	File-element containing the bad data
Insufficient data in file-element	The file-element with insufficient data
Insufficient data in file	End of file (i.e., no associated file-element)
Excess data in file-element	The file-element with excess data

11.4.4.1 Input-Statement. The effect of a prompt-specifier, timeout-expression, and time-inquiry is as described in 10.2. These input-control-items apply only to interactive terminal devices. For other devices and true files, their effect is implementation-defined. The transfer of data with the input-statement also works just as described in 10.2, except that records are treated like input-replies, and end-of-record is treated like end-of-line. Each datum (as defined in 10.1) is assigned in order to a variable in the variable-list. All the INPUT operations (as opposed to the READ operations) are valid only for a file opened as DISPLAY. For any other record-type, an exception results. The input-statement may process several records if the last non-blank character in a record is a comma. If, following a record with a trailing comma, end of file is encountered before all variables have been assigned values, then the remaining variables shall retain their old values, the file pointer shall be positioned to the end of the file, and an exception shall result.

When any of the INPUT statements is executed for a device and a phase 3 exception occurs, implementations may use the recovery procedures specified for true files in this section, or, if an equivalent exception is specified in Section 10, that recovery procedure may be used instead. This is to allow input

from several interactive devices to use the nonfatal recovery procedures.

11.4.4.2 Array-Input-Statement. The array-input-statement behaves just like the input-statement would if the arrays were written out explicitly as array-elements in row major order (the last subscript varying most rapidly). The only additional capability is that of allowing a redim to change the dimensions of the array in accordance with the redim rules for the array-input-statement without a channel-expression (cf. 10.5). Thus, for example, if "DIM A(3)" and OPTION BASE 1 are in effect, the following two statements are equivalent:

```
MAT INPUT #N: A
INPUT #N: A(1), A(2), A(3)
```

The following two statements are also equivalent:

```
MAT INPUT #N: A$(2,2), C(2)
INPUT #N: A$(1,1), A$(1,2), A$(2,1), A$(2,2), C(1), C(2)
```

However, the effect of

```
MAT INPUT #N: A(1), B(A(1))
```

depends on the first datum encountered, since it controls the effective size of array B. Nonetheless, it behaves exactly as would an input-statement for which the appropriate number of array-elements for B had been coded. If an array is encountered whose redim yields a size less than 1 in any dimension, then it, and all subsequent arrays, shall retain their old values, and an exception shall result.

11.4.4.3 Variable-Length-Vectors. The transfer of data and consequent redimensioning of the array of a variable-length-vector takes place just as described in 10.5.

11.4.4.4 Line-Input-Statement. The line-input-statement behaves exactly as described in 10.2, except that records are treated like input-replies, and end-of-record like end-of-line. The content of each successive record is assigned as the value of successive string-variables, including any leading or trailing spaces. A record may contain a null string, and it shall be assigned in the normal way. If end of file is encountered before all variables have been assigned values, then the remaining variables shall retain their old values, the file pointer shall be positioned to end of file, and an exception shall result.

11.4.4.5 Array-Line-Input-Statement. The array-line-input-statement behaves just as would a line-input statement for which the array-elements had been coded out explicitly, instead of as arrays. See semantics in 11.4.4.2 for array-input-statements. Note that here too, the size of a later array may depend on the value assigned to an earlier array, e.g.:

```
MAT LINE INPUT #N: A$(1), B$(VAL(A$(1)))
```

If the first record contained the string " 12 ", then twelve subsequent records would be read into the array B$. If an array is encountered whose redim yields a size less than 1 in either dimension, then it, and all subsequent arrays, shall retain their old values, and an exception shall result.

11.4.4.6 Input-Statements for Channel Zero. Input from channel zero works in accordance with the semantics for non-file devices. For those exceptions for which a different recovery procedure is specified in Section 10 than in Section 11, the procedure of Section 10 shall be used.

11.4.4.7 Read-Operation. The read-statement and array-read-statement are used to retrieve data from files with records of any type. Successive data or values or fields are assigned to successive variables or arrays in the operand list. READ may access several file elements for SEQUENTIAL or STREAM files, but only one for RELATIVE or KEYED files.

11.4.4.8 File Organizations. For non-STREAM files, the variables receive values from the sequence of data or values or fields within a record. There must be just enough data within the record (or records in the case of DISPLAY records with trailing commas) to satisfy the variable-list (except for SKIP REST, see previous text in 11.4.4). For STREAM files, the variables receive their values directly from the sequence that constitutes the file, beginning with the file-element indicated by the pointer, and so file-element boundaries are insignificant. If end of file is encountered before all variables have been assigned values, then the remaining variables shall retain their old values, the file pointer shall be positioned to end of file, and an exception shall result.

11.4.4.9 Display Record-Type. Records in DISPLAY files are sequences of characters. The retrieval of string data shall take place as described in 10.2. Note that retrieving data from a record created with PRINT does not necessarily preserve the same value, since, for instance, leading and trailing spaces are not saved in unquoted strings on input. For numeric data, the

accuracy shall be consistent with the usual semantics for assignment of a numeric-constant to a numeric-variable (i.e., at least six significant digits for OPTION ARITHMETIC NATIVE and at least ten digits for OPTION ARITHMETIC DECIMAL). For a numeric-constant with no more significant digits than the implementation-defined precision, the exact value is assigned with OPTION ARITHMETIC DECIMAL.

A READ operation on a DISPLAY record assigns values exactly as would an INPUT operation with the same variable-list or redim-array-list. Note, however, that not all the INPUT facilities are available for READ, which is record-oriented.

11.4.4.10 Internal Record-Type. An internal record (and a stream file) is a sequence of values. There are two types of value, numeric and string. For INTERNAL file elements, the values shall be retrieved with a variable of the same type as that of the value, otherwise an exception shall result. Thus, the contents of an INTERNAL file element are self-typed. The sequence of values and their types are determined by the record operation that created or modified the file-element(s). When a value is retrieved, the effect is as if the expression with which it was created were assigned to the input variable with a let-statement. The length and content of string values shall be preserved. Numeric values shall also be preserved, consistent with the usual limitations on precision and type associated with the prevailing ARITHMETIC option.

11.4.4.11 Native Record-Type. The TEMPLATE describes the location and type of fields within the record. When reading from a native record, a TEMPLATE shall always be used. It shall not be used with any other record-type. A TEMPLATE is associated with a particular data retrieval statement by means of the template-identifier in the statement, which specifies the template-statement to be used, or a string-expression whose value must be a syntactically correct template-element-list. The string-expression is evaluated before any input takes place and before any redims, substring qualifiers, or subscripts are evaluated. Several statements may use the same template-statement.

When retrieving data from a native record, each variable within the variable-list is associated with a field-specifier within a template; the specifier is then used to return data from a field within a record. This association takes place from left to right, within the variable-list and template-element-list, each variable using the next available field-specifier. A variable is associated with a specifier after a value has been

assigned to the previous variable, and any subscripts, substring-qualifiers, or redims for this variable have been evaluated. If the type of the variable (numeric or string) disagrees with the type of the specifier, an exception results. The number of specifiers shall not be less than the number of variables. Extra specifiers beyond the variables are ignored. The contents of the next field in the record is interpreted according to the specifier, and the resulting value placed in the variable.

When retrieving data, the specifiers of all fields within a record shall be compatible with the specifiers with which they were created, otherwise the results are implementation-defined. In order to be compatible, the creating and retrieving specifiers for a field shall be both of type STRING, with equal string-field-sizes, or both NUMERIC with a numeric-field-size of E, or both NUMERIC with equal integer-sizes and fraction-sizes. An omitted integer-size or fraction-size is treated as equivalent to zero.

When the TEMPLATE specifiers are compatible with the record, then the values are retrieved in accordance with the field sizes. For strings, a value is assigned with length equal to the string-field-size, and contents as originally stored in the record, including any spaces used for padding. For numbers, a value is assigned whose accuracy is limited only by the numeric-field-size and ARITHMETIC option. For numbers stored with a field size of E, or with a fixed-point-size and with no more significant digits than the implementation-defined precision, the exact value is retained under OPTION ARITHMETIC DECIMAL. Otherwise, the numbers are rounded according to the OPTION in effect and stored in the variable. The storing of values in the fields of native records and the effect of field-counts are described in 11.3.4. The only difference upon retrieval is that SKIP specifiers do not generate fields of zero or spaces, but cause the affected fields simply to be skipped over. As before, such specifiers are not associated with variables.

11.4.4.12 Array-Read-Statement. In general, an array-read-statement behaves just like the read-statement would if the arrays were written out explicitly as array-elements. As with INPUT, there is delayed evaluation of redims, and for this reason, when reading from a native record, a variable-field-count is provided. If a fixed-field-count is used, then the number and types of the specifiers shall match the array-elements, as with READ. When the first element of an array is to be associated with the next specifier, however, and if a template-element has just been completed (or if this is the first array in the list), and the next template-element is a variable-field-count, then the

associated specifier is used for all the elements of the array. When the array has been filled, the next array, if any, uses the next template-element, which may or may not also have a variable-field-count. An array shall use either a template-element with a variable-field-count or template-elements with fixed-field-counts, but not both.

11.4.5 Exceptions

The exceptions are grouped according to the phase of processing during which they are detected. Phase 1 exceptions imply no change to the file pointer or variables. Phase 2 exceptions imply no change to the variables. Phase 3 exceptions imply that some variables may have received values from the file.

11.4.5.1 Phase 1 Exceptions

A data retrieval attempts to access an inactive channel (7004, fatal).

An input-, array-input-, line-input-, or array-line-input-statement attempts to access a file opened as INTERNAL or NATIVE (7318, fatal).

The record-setter cannot be processed correctly, as described in 11.2.5 (7002 and 7202-7207, use the procedures of 11.2.5).

A data retrieval statement attempts to access a file opened as OUTPUT (7303, fatal).

The string-expression of a template-identifier is not a syntactically correct template-element-list (8251, fatal).

A template-identifier is used on a file opened as DISPLAY or INTERNAL (7315, fatal).

A read- or array-read-statement does not have a template-identifier when attempting to access a file opened as NATIVE (7316, fatal).

The SKIP REST option is used on a file opened as STREAM (7321, fatal).

11.4.5.2 Phase 2 Exception

For a data retrieval statement, the condition data-found is false and a missing-recovery has not been specified (7305, fatal).

11.4.5.3 Phase 3 Exceptions

An attempt is made to access a record larger than the value of RECSIZE (8302, fatal).

The first index in a redim-bounds is greater than the second (6005, fatal).

A single index used in redim bounds is less than the default lower bound in effect for the program unit (6005, fatal).

The total number of elements required for a redimensioned array exceeds the number of elements reserved by the array's original dimensions (5001, fatal).

A variable or array-element does not agree in type (numeric or string) with its associated TEMPLATE specifier (8252, fatal).

A variable-field-count in a template-element does not coincide with the first element of an array (8253, fatal).

There are not enough TEMPLATE specifiers for all the variables or array-elements (8254, fatal).

A data retrieval statement, other than a line-input-statement or an array-line-input-statement, attempts to access a DISPLAY record that is not a syntactically legal input-reply (8105, fatal).

The datum of a DISPLAY record to be assigned to a numeric variable is not a numeric-constant (8101, fatal).

A value in an INTERNAL record does not agree in type (numeric or string) with the variable to which it is to be assigned (8120, fatal).

A value, datum, or field in a file causes numeric overflow upon assignment to the variable (1008, fatal).

A value, datum, or field in a file causes string overflow upon assignment to a variable (1105, fatal).

There are not enough data, values, or fields within a record of a non-STREAM file for the operands of a data retrieval statement and the record is not DISPLAY with a trailing comma (8012, fatal).

End of file is encountered while seeking further data for the operands of a data retrieval statement (8011, fatal).

There are too many data in a record for the operands of a data retrieval statement and SKIP REST is not specified (8013, fatal).

There is just enough data in a DISPLAY record with a trailing comma to satisfy a request for input, and SKIP REST is not specified (8013, fatal).

11.4.6 Remarks

Implementations may choose to treat underflows as exceptions (1508, nonfatal: supply zero and continue) to permit interception by exception handlers.

11.5 File Data Modification

11.5.1 General Description

Statements are provided to allow the user to modify data previously stored in a file. Such data can either be changed or deleted. The modifications are always done at the record level.

11.5.2 Syntax

Core productions:

None.

Enhanced Files productions:

```
   1. imperative-statement     > rewrite-statement /
                                  array-rewrite-statement /
                                  delete-statement
   2. rewrite-statement        = REWRITE channel-expression
                                  rewrite-control colon
                                  expression-list
   3. array-rewrite-statement = MAT REWRITE channel-expression
                                  rewrite-control colon
                                  array-list
   4. rewrite-control          = (comma rewrite-control-item)*
   5. rewrite-control-item     > missing-recovery / record-setter
   6. delete-statement         = DELETE channel-expression
                                  delete-control
   7. delete-control           = (comma delete-control-item)*
   8. delete-control-item      = missing-recovery / record-setter
  N9. rewrite-control-item     > template-identifier
```

The line-number of a template-identifier shall refer to a template-statement in the same program-unit.

A given rewrite-control-item shall appear at most once in rewrite-control.

A given delete-control-item shall appear at most once in delete-control.

11.5.3 Examples

```
   2. REWRITE #N, KEY = B$, IF MISSING THEN 666: A,B,C$
   3. MAT REWRITE #3, RECORD N-1, WITH 111: X,Y,Z
   6. DELETE #3, KEY "JONES"
```

11.5.4 Semantics

The data modification statements are modeled closely on certain aspects of data retrieval statements and data creation statements. Like the data retrieval statements, they operate on existing records. Like the data creation statements, they can alter the state of a file. The data modification statements are specified only for file-organizations RELATIVE and KEYED. For other file-organizations, their effect is implementation-defined. The data modification statements may be used only with access-mode OUTIN. Except for access-mode, the first and second phase of processing (i.e., checking of file attributes and setting the file pointer) for these statements is exactly like that for the data retrieval statements (cf. 11.4.4), because they operate on existing records. The third phase of processing, undertaken only if the operation is legal, the file pointer successfully set, and data-found is true, is described in 11.5.4.1 through 11.5.4.3 under the individual headings.

11.5.4.1 Rewrite-Statement. The rewrite-statement generates exactly one record, and that record is identical to the one that would be generated by a write-statement with the same expression-list or array-list and template-identifier, if any (cf. 11.3.4), with one exception: for a NATIVE record, fields governed by SKIP are not filled with zero or spaces, but rather the previous contents of the fields are left unchanged. This effect of SKIP occurs only if the TEMPLATE used by the REWRITE is compatible with TEMPLATE last used to alter the record (cf. 11.4.4.11 for the definition of "compatible"). The result of using an incompatible TEMPLATE containing SKIP is implementation-defined. The use of an incompatible TEMPLATE without SKIP is defined as in 11.3.4.11 since the entire record is replaced.

If no exceptions occur during the generation of data to be used for modification of existing data, then the record pointed to by the file pointer is replaced by the record just generated, and the file pointer advanced to the next file-element. This implies that the identifying record-number in a RELATIVE file, or identifying key in a KEYED file is not changed. If there is an exception, the pointer is left as it was set in the second phase (cf. 11.3.4 and 11.4.4) and the data in the file is unchanged.

11.5.4.2 Array-Rewrite-Statement. An array-rewrite-statement behaves just like the rewrite-statement would if the array-elements were written out explicitly. The rules for matching arrays and specifiers in a TEMPLATE are exactly the same as for the array-write-statement (cf. 11.3.4.12).

11.5.4.3 Delete-Statement. The delete-statement causes the record indicated by the file pointer to be deleted, and the file pointer advanced to the next file-element. This implies that for a RELATIVE file, the affected record-area no longer contains a record, and for a KEYED file, the affected record is eliminated from the sequence of records constituting the file.

11.5.5 Exceptions

The following exceptions are grouped according to the phase of processing during which they are detected. Phase 1 exceptions imply no change to the file or file pointer. Phase 2 exceptions imply no change to the file. Phase 3 exceptions also imply no change to the file.

11.5.5.1 Phase 1 Exceptions

A data modification statement attempts to access an inactive channel (7004, fatal).

A data modification statement attempts to access channel zero (7320, fatal).

The record-setter cannot be processed correctly, as described in 11.2.5 (7002 and 7202-7207, use the procedures of 11.2.5).

A data modification statement attempts to access a file opened as INPUT or as OUTPUT (7322, fatal).

The string-expression of a template-identifier is not a syntactically correct template-element-list (8251, fatal).

A template-identifier is used on a file opened as INTERNAL or DISPLAY (7315, fatal).

A rewrite- or array-rewrite-statement does not have a template-identifier when attempting to access a file opened as NATIVE (7316, fatal).

11.5.5.2 Phase 2 Exception

For a data modification statement, the condition data-found is false, and a missing-recovery has not been specified (7305, fatal).

11.5.5.3 Phase 3 Exceptions

An attempt is made to rewrite a record larger than the value of RECSIZE (8301, fatal).

An expression or array-element does not agree in type (numeric or string) with its associated TEMPLATE specifier (8252, fatal).

A template-element with a variable-field-count does not coincide with the first element of an array (8253, fatal).

There are not enough TEMPLATE specifiers for all the expressions or array-elements (8254, fatal).

A numeric value has significant digits to the left of the available digit places in the field of a template (8255, fatal).

A string value is longer than the length of its field in the template (8256, fatal).

11.5.6 Remarks

Note that DELETE and REWRITE will affect the record indicated by the file pointer, even if the pointer is set with NEXT or left as is from a previous operation (i.e., if the record-setter is absent).

12. Exception Handling and Debugging

12.1 Exception Handling

12.1.1 General Description

Exception handling facilities provide a means of regaining control of a program after an exception has occurred.

12.1.2 Syntax

```
 1. protection-block           = when-use-block /
                                 when-use-name-block
 2. when-use-block             = when-line when-block
                                 use-line exception-handler
                                 end-when-line
 3. when-line                  = line-number WHEN EXCEPTION IN
                                 tail
 4. when-block                 = block*
 5. use-line                   = line-number USE tail
 6. exception-handler          = block*
 7. end-when-line              = line-number END WHEN tail
 8. when-use-name-block        = when-use-name-line
                                 when-block end-when-line
 9. when-use-name-line         = line-number WHEN EXCEPTION USE
                                 handler-name tail
10. handler-name               = routine-identifier
11. handler-return-statement   = RETRY / CONTINUE
12. exit-handler-statement     = EXIT HANDLER
13. cause-statement            = CAUSE EXCEPTION exception-type
14. exception-type             = index
15. detached-handler           = handler-line exception-handler
                                 end-handler-line
16. handler-line               = line-number HANDLER
                                 handler-name tail
17. end-handler-line           = line-number END HANDLER
                                 tail
18. numeric-supplied-function  > EXLINE / EXTYPE
19. string-supplied-function   > EXTEXT dollar-sign
```

Handler-return-statements and exit-handler-statements shall occur only within exception-handlers. The no-argument numeric-supplied-functions EXLINE and EXTYPE shall be invoked only within exception-handlers. EXTEXT$ takes a single numeric argument, which is an index.

No line-number in a control-transfer outside a protection-block shall refer to a line in that protection-block other than

its when-line or when-use-name-line. No line-number in a control-transfer inside an exception-handler shall refer to a line outside that exception-handler other than its own end-handler-line or end-when-line, nor shall a line-number in a control-transfer outside an exception-handler refer to a line inside that exception-handler or to its end-handler-line or end-when-line.

A detached-handler referred to in a when-use-name-line within an internal-proc-def shall be defined in the same internal-proc-def. A detached-handler referred to in a when-use-name-line that is not within an internal-proc-def shall be defined in the same program-unit but not within an internal-proc-def. No two handler-lines in the same program unit shall have the same handler-name. A detached-handler shall not appear within a protection-block.

A protection-block shall not appear within an exception-handler.

12.1.3 Examples

1. Example 1: Handling errors in input-replies by allowing the input-reply to be resupplied after issuing a suitable message

```
100 WHEN EXCEPTION IN
110     PRINT "Enter your age and weight"
120     INPUT a, w
130     IF  a > 10 THEN
140         PRINT "What is your height"
150         INPUT h
160     END IF
170 USE
180     PRINT "Please enter numbers only"
190     RETRY
200 END WHEN
```

Example 2: Dynamic file opening

```
100 HANDLER file_trouble
110    LET file_ok$ = "false"
120    IF EXTYPE = 7107 THEN
130       LET message$ = " doesn't exist"
140    ELSEIF EXTYPE = 7102 THEN
150       LET message$ = " is the wrong type"
160    ELSE
170       LET message$ = " couldn't be used"
```

```
 180     END IF
 190     PRINT "file "; filename$; message$; " try again"
 200 END HANDLER

 500 DO
 510     INPUT filename$
 520     LET file_ok$ = "true"
 530     WHEN EXCEPTION USE file_trouble
 540        OPEN #n: NAME filename$  ! other parameters omitted
 550     END WHEN
 560 LOOP UNTIL file_ok$ = "true"
```

Example 3: Nested handlers

```
 100 WHEN EXCEPTION IN
 110     DO
 120        READ #1, IF MISSING THEN EXIT DO: A
 130        LET I = I+1                ! I initialized outside loop
 140        WHEN EXCEPTION IN
 150           LET B(I) = 1000*A*A
 160        USE
 170           ! Assume it is numeric overflow
 180           LET B(I) = MAXNUM
 190           CONTINUE
 200        END WHEN
 210     LOOP
 220 USE
 230     IF EXTYPE = 8101 THEN        ! non-numeric data
 240        RETRY                     ! get next data item
 250     ELSE                         ! give up
 260        PRINT "Unable to process file"
 270        STOP
 280     END IF
 290 END WHEN
```

13. CAUSE EXCEPTION I

12.1.4 Semantics

When an exception occurs during the execution of a program-unit, the action taken shall depend upon whether or not the exception occurs within a when-block. If the exception occurs outside a when-block, then the default exception handling procedures specified in this standard shall be applied (cf. 2.4). If the exception occurs within a when-block, then the default exception handling procedures, which require that an exception be reported, shall not be applied; instead, control shall be

transferred to the exception-handler associated with the inner-most protection-block within which the exception occurred.

When the protection-block is a when-use-block, the associated exception-handler is that which follows the use-line of the protection-block. When the protection-block is a when-use-name-block, the associated exception-handler is the detached-handler named in the when-use-name-line of the protection-block. In all respects, a detached-handler behaves semantically as though it were an exception-handler in the when-use-block of the when-block with the exception.

Within an exception-handler, the type of the exception that caused that handler to be executed shall be obtainable as the value of the parameterless function EXTYPE. The values of EXTYPE for all exceptions defined in this standard are specified in Table 9, along with the description of each exception in this standard. The line-number of the line whose execution caused the exception shall be obtainable as the value of the parameterless function EXLINE.

There are four means of exiting from an exception-handler.

(1) Execution of the handler-return-statement CONTINUE shall cause control to be transferred to the statement lexically following that which caused the exception. If the exception occurred in a line that begins or is part of a structure (such as a do-line, loop-line, for-line, if-then-line, elseif-then-line, select-line, or case-line), then control shall be transferred to the statement lexically following the entire structure of which the line is a part.

(2) Execution of the handler-return-statement RETRY shall cause control to be transferred to the statement or line that caused the exception, causing the statement or line to be re-executed; if that statement was performing data retrieval, then the previous input-reply or line-input-reply shall be discarded and a new one requested.

(3) If control reaches an end-when-line that terminates an exception-handler or reaches an end-handler-line, then control shall be transferred to the line following the end-when-line of the protection-block within which the exception occurred with no further effect.

(4) Execution of an exit-handler-statement shall cause the exception to be propagated to the lexical environment surrounding the innermost protection-block containing the exception (also

note the effect of calls and function invocations described later in this subsection). That is, the effect on handling the exception is as if the exception-handler did not exist (except for the effect of any statements already executed in the handler), and the rules for handling the original exception depend upon whether or not the exception occurs within some outer when-block.

If execution reaches a use-line in a when-use-block, or an end-when-line in a when-use-name-block, then control shall be transferred to the line following the protection-block of which the use-line or end-when-line is a part. If execution reaches an end-handler-line of a detached-handler, control shall continue at the line following the end-when-line of the when-use-name-block causing the exception. If execution reaches a handler-line of a detached-handler other than by the occurrence of an exception, control shall then continue at the line immediately following the end-handler-line.

A separate GOSUB stack is associated with each exception-handler (cf. 8.2) so RETURN never attempts to transfer control into or out of an exception handler.

Execution of a cause-statement shall result in the occurrence of a fatal exception and the setting of EXTYPE to the rounded value of the exception-type.

If an exception is caused by a statement lexically within an exception-handler, then this new exception shall be handled by the default exception-handling procedures.

If a fatal exception occurs in a procedure-part or internal-proc-def and either:

(1) The line causing the fatal exception is not contained in a when-block and therefore no exception-handler is entered, or

(2) An exception-handler is entered, an exit-handler-statement is executed with the handler, and there is no lexically surrounding when-block to intercept the exception,

then the fatal exception shall be propagated back to the line that invoked the procedure-part or internal-proc-def. This propagation shall continue to occur until either:

(1) A user-defined exception-handler resolves the exception by execution of a handler-return-statement or by causing control

to pass to an end-handler-line or to an end-when-line that terminates the exception-handler, or

(2) The main-program or a parallel-section is reached, in which case the default exception-handling procedures are applied.

If an exception-handler is invoked as a result of this process, then the value returned by the EXTYPE function shall be 100000 plus the value that would have been returned by EXTYPE in the procedure-part or internal-proc-def in which the exception originally occurred. The value of EXLINE shall be the line-number of the most recent line to which the exception was propagated (i.e., the line lexically within the when-block associated with the exception-handler, not the line of the original exception).

The default exception-handling procedures shall always report the EXTYPE and EXLINE of the original exception.

The value of EXTYPE for exceptions defined by local enhancements to this standard shall be negative. When negative values of EXTYPE are propagated, the value shall be -100000 plus the value that would have been returned by EXTYPE for the original exception.

Values of EXTYPE from 1 to 999 will not be used by future enhancements to this standard, nor shall they be used by local enhancements to this standard.

The value of EXTEXT$ shall be the text part of the error message provided by the system for the exception number obtained by rounding its argument to an integer. If its argument is not the exception number of a standard system exception, the value of EXTEXT$ shall be the null string.

If the main-program is reached and no exception-handler is invoked there as a result of the original exception, then the exception shall be handled by the default exception handling procedures specified in this standard.

12.1.5 Exception

A cause-statement is executed (exception-type, fatal).

12.1.6 Remarks

Users should note that there are two kinds of exception propagation specified in this . First, there is "lexical"

propagation, outward to surrounding protection-blocks within a program-unit or internal-proc-def. If this process propagates the exception outside of any such protection-block, "invocation" propagation takes effect, passing the exception back to invoking statements.

The function EXLINE should be used with caution, as the use of editing facilities that renumber lines in a program (cf. 16.2) may invalidate computations involving EXLINE. For example, the program fragment

```
1000 SELECT CASE INT(EXLINE/100)
1010 CASE 1, 2
       ...
1100 CASE 3 TO 7
       ...
```

would probably behave differently if lines 100 through 800 were renumbered.

When a fatal exception is propagated back to invoking statements and the default exception-handling procedure is applied as a result, only the original exception's EXTYPE and EXLINE must be reported. Implementations may, however, also report the line-numbers of the lines through which the exception was propagated, or any other information deemed useful.

It is not possible to pass a nonfatal exception back to a calling routine since it will be handled either by an exception-handler in the called routine or by the system handler. An exception handler may, however, cause a fatal exception with a cause-statement.

The cause-statement is not intended actually to simulate any given exception, but rather to raise a fatal exception with a specified value of EXTYPE. In particular, if the specified EXTYPE is the same as for some nonfatal exception, implementations need not apply the recovery procedure as though that nonfatal exception had actually occurred. It is presumed that a program will normally contain an exception-handler to receive and process the exception.

All positive values of EXTYPE are reserved for future versions of this standard. Exceptions defined by local enhancements to this standard should be identified by negative values for EXTYPE, following the categories established in Table 9. The value returned by EXTYPE for an exception defined in a local enhancement and occurring in a procedure-part or internal-

proc-def should be -100000 plus the negative value identifying that exception. For example, if an implementation chose an EXTYPE value of -4029 for an invalid argument in a new built-in function, and if that exception occurred in a subprogram, but was not handled there, then the value of EXTYPE in an exception-handler in a calling program should be -104029.

It is recommended that implementations use the "zero-th" value in a class of EXTYPE values to represent "other exceptions of this type." For example, an EXTYPE value of 1000 might represent all overflows not defined in this standard.

Values of EXTYPE from 1 to 999 may only occur from cause-statements in application programs. These values should be encouraged for use, since they will not be assigned standard meanings in future enhancements to this standard.

CONTINUE should be used with caution. For instance, if an exception occurs within a def-statement, on-gosub-statement, on-goto-statement, or if-statement, CONTINUE will transfer control to the lexically following line. Such action may not be equivalent to resumption of normal flow of control.

The following example illustrates the effect of CONTINUE with control structures:

```
100 WHEN EXCEPTION IN
120       INPUT PROMPT "Enter your age and weight ": a, w
130       DO WHILE a > 1
140          IF a < 9999999999 THEN
150             INPUT PROMPT "What is your height ": h
160             PRINT "Check the following:"
170             PRINT "Age:"; a, "Weight:"; w, "Height:"; h
200             INPUT PROMPT "Enter your age ": a
210          END IF
220          PRINT "Lexically following IF"
230       LOOP
240       PRINT "Lexically following DO WHILE"
...
```

For exception in line:	CONTINUE transfers control to line:
120	130
130	240
140	220
150	160

The precise format of the values of the EXTEXT$ function is implementation-defined. In particular, implementations may choose to omit, or to mark in a special way, those fields in an error message that are specific to a particular instance of an exception, such as the line number at which the exception occurred or the value of an out-of-range subscript.

12.2 Debugging

12.2.1 General Description

Debugging facilities are provided by language statements in order to allow test points to be built into a program. These statements allow the user to set break points, to trace the action of the program, and to turn the debugging system on and off within each program-unit.

12.2.2 Syntax

```
1.  debug-statement   =  DEBUG ( ON / OFF )
2.  break-statement   =  BREAK
3.  trace-statement   =  TRACE ON (TO channel-expression)? /
                         TRACE OFF
```

12.2.3 Examples

```
3. TRACE ON
   TRACE ON TO #3
```

12.2.4 Semantics

Each program-unit shall have a debugging status, which is either active or inactive at any given time. The debugging status of a program-unit shall persist between invocations of that program-unit (with the exception of the main program). Changes in the debugging status of one program-unit shall not affect the debugging status of any other program-unit. At the beginning of execution of the program, debugging shall be inactive for all program-units.

Execution of the debug-statement DEBUG ON shall cause debugging to become active for the program-unit in which that debug-statement occurs. Debugging shall remain active for the remainder of that invocation of that program-unit, and for each subsequent invocation of that program-unit, until the debug-statement DEBUG OFF is executed in that program-unit. Execution of the debug-statement DEBUG OFF shall cause debugging to become inactive for the remainder of that invocation of that program-unit, and for each subsequent invocation of that program-unit, until the debug-statement DEBUG ON is executed in that program-unit.

The execution of a break-statement when debugging is active shall cause an exception. The standard recovery procedure from this exception shall be to report the line-number of the break-

statement and to signify to the user that interaction with the debugging system is possible. The actions allowed by the debugging system, including the method for continuing execution or terminating execution of the program, are implementation-defined. If the execution of a program reaches a line containing a break-statement, and debugging is inactive, then it shall proceed to the next line with no other effect.

The execution of a trace-statement when debugging is active shall turn tracing on (if ON is specified) or off (if OFF is specified) in the program-unit containing the trace-statement. Prior to the execution of any trace-statement upon each separate entry to a program-unit, tracing shall be off. If the execution of a program reaches a line containing a trace-statement, and debugging is inactive, then it shall proceed to the next line with no other effect.

The execution of a trace-statement shall not affect the debugging status, nor shall the execution of a debug-statement affect the tracing status (ON or OFF).

Whenever tracing is on and debugging is active in a program-unit, the following actions shall occur each time a line of the specified type is executed:

(1) For any line that interrupts the sequential order of execution of lines in a program, both the line-number of that line and the line-number of the next line to be executed shall be reported

(2) For any line that assigns a value to a variable or to an element of an array, both the line-number of that line and any values assigned by execution of that line shall be reported.

Whenever tracing has been turned on via a trace-statement with a channel-expression, trace reports shall be directed to the (display format) file assigned to the specified channel. If no channel-expression has been specified, the trace report shall be directed to the device associated with channel zero.

The contents of the trace report are implementation-defined, but shall include at least the name of the variable traced, as that name lexically appears in the statement causing the trace report, and its value; if the variable is an array element, the value(s) of its subscripts shall also be included.

12.2.5 Exceptions

A break-statement is executed when debugging is active (10007, nonfatal: the recovery procedure is to report the line-number of the statement and to permit interaction with the debugging system).

An attempt is made to direct a trace report to an inactive channel (7401, fatal).

An attempt is made to direct a trace report to a file that is not display format opened with access OUTPUT or OUTIN (7402, fatal).

12.2.6 Remarks

Since an array-assignment assigns a value to each element of an array, tracing an array-assignment causes reporting of all new array element values.

The form of all trace reports is implementation-defined.

Implementations may provide debugging facilities through commands in addition to statements. It is recommended that such commands use the same keywords as the statements.

13. Graphics

The facilities provided in 13.1 through 13.4 are a subset of those provided by level 0b of the Graphical Kernel System (GKS) as defined in ANSI X3.124-1985 and ISO 7942-1985. The values of the EXTYPE function for exceptions defined in GKS are 11000 plus the value of the GKS error number. Suggested syntax for implementations wishing to extend graphics to full level 0b of GKS is given in Appendix F. Extensions to the American National Standard and the International Standard for GKS are provided in 13.5.

In GKS terms, any Basic program that includes statements from Section 13 of this standard has implied calls to the functions OPEN GKS, OPEN WORKSTATION(#0,"Maindev",1), and ACTIVATE WORKSTATION #0 before any graphics statements are executed, and calls to the functions DEACTIVATE WORKSTATION #0, CLOSE WORKSTATION #0, and CLOSE GKS as the program terminates.

13.1 Coordinate Systems

13.1.1 General Description

The coordinates used to produce graphic output may be chosen to suit the application. The range of this system of "problem coordinates" (world coordinates) is established by a SET WINDOW statement. This range is mapped into a rectangular portion of an abstract viewing surface, which can be specified by a SET VIEWPORT statement. It is possible to specify what part of this abstract viewing surface will be presented to the user on the display surface by a SET DEVICE WINDOW statement. This rectangle, in turn, may be located on the display surface by a SET DEVICE VIEWPORT statement.

No output will be produced outside the device viewport. It is possible to guarantee that all graphic output that lies outside the viewport will be eliminated by enabling clipping.

Ask-statements are provided to determine the current values for the parameters established by execution of one of the set-statements or by default.

13.1.2 Syntax

```
1. set-object          > WINDOW boundaries /
                         VIEWPORT boundaries /
                         DEVICE WINDOW boundaries /
                         DEVICE VIEWPORT boundaries /
                         CLIP string-expression
2. boundaries          = boundary comma boundary comma
                         boundary comma boundary
3. boundary            = numeric-expression
4. ask-statement       > ASK ask-object status-clause?
5. status-clause       = STATUS numeric-variable
6. ask-object          > WINDOW boundary-variables /
                         VIEWPORT boundary-variables /
                         DEVICE WINDOW boundary-variables /
                         DEVICE VIEWPORT boundary-variables /
                         DEVICE SIZE numeric-variable comma
                         numeric-variable comma
                         string-variable /
                         CLIP string-variable
7. boundary-variables  = numeric-variable comma
                         numeric-variable comma
                         numeric-variable comma
                         numeric-variable
```

13.1.3 Examples

```
1.  WINDOW 0, PI*2, -1, 1
    Viewport .5*width, width, .5*height, height
    DEVICE WINDOW 0, .8, 0, 1
    DEVICE VIEWPORT .3, .5, .1, 1
    CLIP "Off"

4.  ASK WINDOW X1, X2, Y1, Y2
    ASK VIEWPORT L, R, B, T
    ASK DEVICE WINDOW XMIN, XMAX, YMIN, YMAX
    ASK DEVICE VIEWPORT LEFT, RIGHT, BOTTOM, TOP
    Ask device size Width, Height, Units$
    ASK CLIP CLIP_STATE$
```

13.1.4 Semantics

Graphic output is specified in problem coordinates. A normalization transformation defines the mapping from the problem coordinate system onto the normalized device coordinate (NDC) space, which can be regarded as an abstract viewing surface.

The normalization transformation is specified by defining the limits of a rectangular area, called a window, in problem coordinates. The window is mapped linearly onto a specified rectangular area, called a viewport, in NDC space.

Execution of a set-statement with the keyword WINDOW shall establish the boundaries of the window. The parameters represent the problem coordinates of the left, right, bottom, and top edges, in that order, of the window rectangle. At the start of program execution the window values are (0,1,0,1).

Execution of a set-statement with the keyword VIEWPORT shall establish the viewport boundaries. The parameters represent the normalized device coordinates of the left, right, bottom, and top edges, in that order, of the viewport rectangle. Viewport coordinates shall not be less than zero nor more than one. The value of the left coordinate shall be less than the right, and the bottom less than the top. At the start of program execution, the viewport values are (0,1,0,1).

The viewport may also be used to define a clipping rectangle. Execution of a set-statement with the keyword CLIP shall enable or disable clipping to the viewport boundary (cf. 13.3) depending on whether the value of the string-expression is "ON" or "OFF". The letters in the value of the string-expression may be any combination of uppercase and lowercase. At the start of program execution, clipping shall be enabled.

A device transformation is used to map a rectangle in NDC space called a device window uniformly onto a rectangle on a physical surface called a device viewport. This transformation shall perform equal scaling with a positive scale for both axes. To ensure equal scaling, the device transformation maps the device window onto the largest rectangle that can fit within the device viewport such that the aspect ratio of the device window is preserved and the lower-left corner of the device window is mapped onto the lower-left corner of the device viewport.

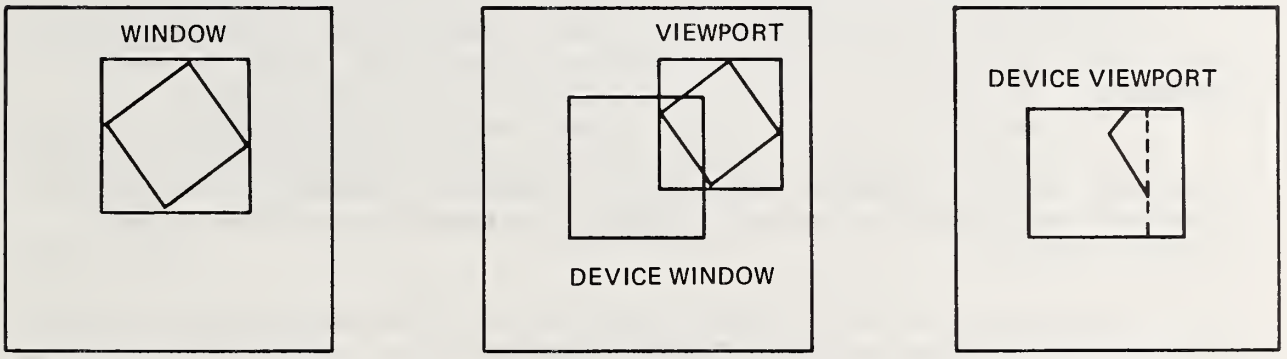
Execution of a set-statement with the keywords DEVICE WINDOW shall establish the boundaries of the device window. The parameters represent the normalized device coordinates of the left, right, bottom, and top edges, in that order, of the device window rectangle. These coordinates shall not be less than zero nor greater than one. The value of the left coordinate shall be less than the right, and the bottom less than the top. At the start of program execution, the device window values are (0,1,0,1). To ensure that no output outside the device window is displayed, clipping takes place at the device window boundaries.

This clipping may not be disabled. Execution of a set-statement with the keywords DEVICE WINDOW shall cause the display surface to be cleared if it is not already clear.

Execution of a set-statement with the keywords DEVICE VIEWPORT shall establish the boundaries of the device viewport. The parameters represent the coordinates of the left, right, bottom, and top edges, in that order of the device viewport rectangle. Units for the device viewport shall be meters on a device capable of producing a precisely scaled image and appropriate device dependent coordinates otherwise. The left and bottom edges of a display surface are represented by the coordinate value zero. At the start of program execution, the device viewport is the entire screen. Execution of a set-statement with the keywords DEVICE VIEWPORT shall cause the display surface to be cleared if it is not already clear.

Figure 1 illustrates the relationship between the the window, the viewport, the device window, and the device viewport; clipping is assumed "ON".

Figure 1. Relationships of Windows and Viewports



If a status-clause is included in an ask-statement, a status associated with the execution of the ask-statement shall be returned in the numeric-variable. If the statement returned meaningful values for the ask-object, a value of zero shall be returned in the status-clause. If the ask-statement could not return meaningful values for the ask-object, a nonzero value shall be returned in the status-clause that is defined with the semantics of the particular ask-object. If an ask-statement with a particular ask-object is always expected to return meaningful values, the semantics for that ask-object do not specify alternate status values and zero shall always be returned.

Execution of an ask-statement with one of the keywords WINDOW, VIEWPORT, DEVICE WINDOW, or DEVICE VIEWPORT shall provide the current values for the specified rectangle. Values for the left, right, bottom, and top sides, respectively, shall be assigned to the boundary-variables equal to the values last established by a set-statement, or, if no appropriate set-statement has been executed, equal to the default value.

Execution of an ask-statement with the keywords DEVICE SIZE shall assign to the first numeric variable the size in the horizontal direction and shall assign to the second numeric variable the size in the vertical direction of the available display surface. The string-variable shall be assigned the value "METERS" if the sizes are in meters or the value "OTHER" if the units of measure are device coordinates or other units. The values "METERS" and "OTHER" shall consist of upper-case-letters.

Execution of an ask-statement with the keyword CLIP shall assign the value "ON" to the string-variable if clipping is enabled and the value "OFF" if it is disabled. The values returned shall be all upper-case-letters.

13.1.5 Exceptions

The boundaries in a set-statement specify a rectangle of zero width or height (11051, nonfatal: continue with current values).

The boundaries in a set-statement with the keywords VIEWPORT, DEVICE WINDOW, or DEVICE VIEWPORT specify a rectangle of negative width or height (11051, nonfatal: continue with current values).

A boundary of the viewport is not in the range [0,1] (11052, nonfatal: continue with current values).

A boundary of the device window is not in the range [0,1] (11053, nonfatal: continue with current values).

A boundary of the device viewport is not in the display space (11054, nonfatal: continue with current values).

The value of the string-expression in a set-statement with the keyword CLIP is neither "ON" nor "OFF" after conversion to upper-case (4101, nonfatal: continue with current value).

13.1.6 Remarks

The manner in which a particular graphic display device is selected by a program is implementation-defined.

The meaning of a window with the left edge greater than the right or the bottom edge greater than the top is implementation-defined. If possible, implementations should provide appropriately inverted images. The effect of all graphic input and output is defined in terms of the abstract problem space, in which lower values are to the left and down, and higher values to the right and up. When this problem space is mapped to NDC, it may be inverted as indicated by the order of the WINDOW boundaries. This relaxes the GKS rule that states that reversal window coordinates causes an error.

SET WINDOW, SET VIEWPORT, SET DEVICE WINDOW, and SET DEVICE VIEWPORT correspond to the GKS functions SET WINDOW, SET VIEWPORT, SET WORKSTATION WINDOW, and SET WORKSTATION VIEWPORT, respectively. The GKS transformation number is one in these statements. The GKS workstation number is #0 in these statements.

SET CLIP corresponds to the GKS function SET CLIPPING INDICATOR.

ASK WINDOW and ASK VIEWPORT correspond to the GKS function INQUIRE NORMALIZATION TRANSFORMATION for normalization transformation one.

ASK CLIP corresponds to the GKS function INQUIRE CLIPPING INDICATOR.

ASK DEVICE WINDOW and ASK DEVICE VIEWPORT correspond to the current workstation window and current workstation viewport parameters, respctively, of the GKS function INQUIRE WORKSTATION TRANSFORMATION with a workstation identifier of one.

ASK DEVICE VIEWPORT before any SET DEVICE VIEWPORT may be used to find the device coordinates of the full available device surface.

ASK DEVICE SIZE corresponds to the device coordinate units and maximum display surface size in device coordinate units parameters of the GKS function INQUIRE MAXIMUM DISPLAY SURFACE SIZE.

Many of the ask-objects defined in Appendix F have cases where no meaningful values can be returned. In these cases, the value returned in a status-clause shall be 11000 plus the error indicator parameter value specified by GKS (ANSI X3.124-1985 and ISO 7942-1985).

13.2 Attributes and Screen Control

13.2.1 General Description

A graphic display device may possess several styles of lines or points, each with a particular width or texture. A particular style may be selected for graphic output. A graphic device also may be able to draw lines or fill areas, or do both, in a variety of colors. Particular colors may be selected for line drawing and screen background.

The current style and color of the geometric object may be determined by ask-statements. The number of colors and the number of line or point styles available may also be determined by ask-statements.

A graphic display device may possess several varieties of text, each with a particular height or orientation. A combination may be selected for text output.

The clear-statement clears the entire screen, returning it to its background color. For hard-copy devices, the clear-statement causes the paper to advance, the pen to move aside, or similar action.

13.2.2 Syntax

```
1. imperative-statement  > clear-statement
2. clear-statement       = CLEAR
3. set-object            > primitive-1 STYLE index /
                           primitive-2 COLOR index /
                           TEXT text-facet numeric-expression /
                           TEXT JUSTIFY string-expression
                           comma string-expression /
                           COLOR MIX left-parenthesis index
                           right-parenthesis rgb-list
4. primitive-2           = primitive-1 / TEXT / AREA
5. primitive-1           = POINT / LINE
6. rgb-list              = numeric-expression comma
                           numeric-expression comma
                           numeric-expression
7. ask-object            > primitive-1 STYLE numeric-variable /
                           primitive-2 COLOR numeric-variable /
                           TEXT text-facet numeric-variable /
                           TEXT JUSTIFY string-variable comma
                           string-variable /
                           MAX primitive-1 STYLE
                           numeric-variable /
```

```
                              MAX COLOR numeric-variable /
                              COLOR MIX left-parenthesis index
                              right-parenthesis mix-list
  8. mix-list               = numeric-variable comma
                              numeric-variable comma
                              numeric-variable
  9. text-facet             = HEIGHT / ANGLE
```

13.2.3 Examples

```
  3. LINE STYLE 2
     TEXT COLOR 5
     AREA COLOR RED
     TEXT HEIGHT (N+3)/42
     Text justify "Center", "Half"
     color mix (4) .5, R*.5, .3
  7. Point style P_style
     Max color color_max
     Max point style PtStyles
```

13.2.4 Semantics

Execution of a clear-statement shall clear the graphic display if not already clear. For soft-copy devices, it shall erase the screen. For hard-copy devices, it shall advance the medium or allow the device operator to change it.

Execution of a set-statement with the keywords LINE STYLE or POINT STYLE shall cause the index to be evaluated by rounding to obtain an integer N and shall establish the style for subsequent lines or points to be the Nth one of the set of available line or point styles. The number of line styles available is implementation-defined, but shall be at least four. A line style of one shall correspond to a drawing of solid lines, a line style of two to dashed lines, a line style of three to dotted lines, and a line style of four to dashed-dotted lines. All other values for line style are implementation-defined. At the initiation of program execution, the line style shall be one.

Point styles produce centered symbols. The number of point styles is implementation-defined, but shall be at least five. A point style of one shall correspond to a dot (.), a point style of two to a plus sign (+), a point style of three to an asterisk (*), a point style of four to a circle (o), and a point style of five to an x (x). All other values for point-style are implementation-defined. At the start of program execution, the point style shall be three.

Execution of an ask-statement with the keywords LINE STYLE or POINT STYLE shall assign the number of the actual current line style or point style to the numeric-variable.

Execution of an ask-statement with the keywords MAX LINE STYLE or MAX POINT STYLE shall assign to the numeric-variable the largest value of LINE STYLE or POINT STYLE, respectively, available.

All values for style shall be valid from one to the number returned by ASK MAX POINT STYLE or ASK MAX LINE STYLE.

Execution of a set-statement with one of the keyword pairs POINT COLOR, LINE COLOR, TEXT COLOR, or AREA COLOR shall cause the index to be evaluated by rounding to obtain an integer N and shall establish the color index of subsequent points, lines, text, or filled areas to be the Nth one of the set of colors, if possible with the current graphics device. This color is called a foreground color. At the initiation of execution, the color associated with each index is implementation-defined, and the foreground color indices shall all have the value one.

Execution of an ask-statement with one of the keyword pairs POINT COLOR, LINE COLOR, TEXT COLOR, or AREA COLOR shall assign to the numeric-variable the current value of the color index for points, lines, text or filled areas, as appropriate.

Execution of a set-statement with the keywords COLOR MIX shall cause the index to be rounded to obtain an integer and shall establish the color associated with the index. The three following numeric-expressions shall establish the intensities of the colors red, green, and blue, respectively, associated with the specified color index. The values for red, green, and blue shall be greater than or equal to zero and less than or equal to one.

The color index zero shall represent the background color on devices for which that is meaningful. It is implementation-defined whether the effect of SET COLOR MIX changes already-displayed colors or only subsequently-displayed colors.

Execution of an ask-statement with the keywords COLOR MIX shall cause the index to be rounded to obtain an integer. The numeric-variables shall be assigned the intensities of red, green, and blue, in that order, associated with the specified color index. The values returned shall be those set by the last set-statement executed with the keywords COLOR MIX and the same index. If it was not possible to precisely set the color mix,

the values returned shall be those actually in force. The values of the color mix at the start of program execution shall be implementation-defined. When an ask-statement with the keywords COLOR MIX and a status-clause is executed, the value returned in the status-clause shall be 11086 if the color index is less than zero or greater than the maximum available color index, or shall be 11087 if no color mix has been established for the index. In these cases, values of zero shall be returned for red, green, and blue.

Execution of an ask-statement with the keywords MAX COLOR shall assign to the numeric-variable the largest distinct value available as an index for SET COLOR MIX, SET POINT COLOR, SET LINE COLOR, SET TEXT COLOR, or SET AREA COLOR. All values for color index from zero to this value should be valid.

13.2.4.1 Text Attributes. Execution of a set-statement with the keywords TEXT HEIGHT sets the approximate height, in problem coordinates, of characters printed by subsequent graphic-text-statements. After current viewing transformations map the desired size of a character to device coordinates, the largest hardware character size that does not exceed the desired height is selected. If all available sizes exceed the desired height, the smallest hardware character set, if any, shall be used. If no hardware character set exists, software-generated characters that obey the above rules shall be used. The default value of text height is 0.01. Text height is the height of upper-case-letters.

Execution of an ask-statement with the keywords TEXT HEIGHT shall assign to the numeric-variable the current value of text height. If the actual text height is different from that set in the most recent set-statement with the keywords TEXT HEIGHT, the actual text height shall be returned.

A set-statement with the keywords TEXT ANGLE shall establish the angle in problem coordinates at which subsequent text shall be displayed. For a TEXT ANGLE of zero, the label is drawn on the screen with the normal horizontal orientation, the first character being the leftmost. For a nonzero TEXT ANGLE, the label is rotated from this orientation by the amount designated (degrees or radians, according to the prevailing ANGLE option) in a counterclockwise direction, using the JUSTIFY point as the pivot. Implementations shall display strings of characters horizontally for angles that are integer multiples of 180 degrees, and vertically for angles that are odd multiples of 90 degrees. Text for angles that are odd multiples of 45 degrees shall be neither horizontal nor vertical, but some intermediate

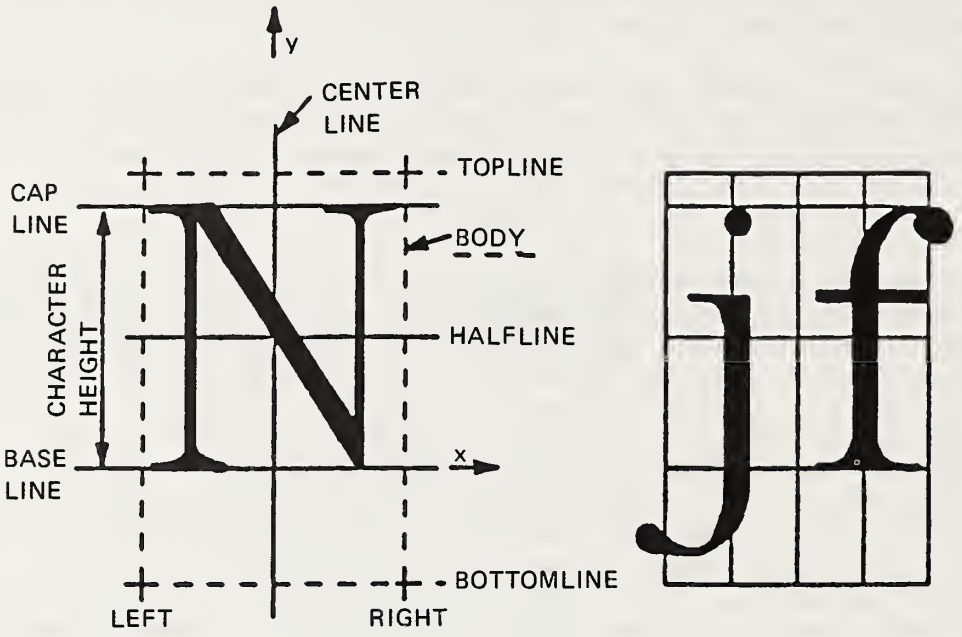
diagonal direction. The availability of additional orientations shall be implementation-defined. The orientation of individual characters in a label shall be implementation-defined. At the start of execution, the text angle shall be 0.

Execution of an ask-statement with the keywords TEXT ANGLE shall assign to the numeric-variable the current value of text angle. If the actual text angle is different from that set in the most recent set statement with the keywords TEXT ANGLE, the actual text angle shall be returned.

Execution of a set-statement with the keywords TEXT JUSTIFY shall evaluate the string-expressions to establish the position of a rectangle surrounding subsequent text output, relative to the text position, which is specified by initial-point. The valid values of the first string are "LEFT," "CENTER," or "RIGHT" and specify the horizontal component of text justification. The valid values of the second string are "TOP," "CAP," "HALF," "BASE," or "BOTTOM" and establish the vertical component of text justification. The above values may be any mixture of upper-case- and lower-case-letters. A horizontal component of "LEFT" shall correspond to the left side of the text rectangle passing through the text position. A value of "CENTER" shall correspond to the text position lying midway between the left and right sides of the text extent rectangle. A value of "RIGHT" shall correspond to the right side of the text rectangle passing through the text position. The vertical component corresponds to one of the font specific lines in the definition of a character in the accompanying figure. A value of "TOP" causes the top of the text extent rectangle to pass through the text position. A value of "CAP" causes the text position to lie on the capline of the whole string. A value of "HALF" causes the text position to lie on the halfline of the the whole string. A value of "BASE" causes the text position to lie on the baseline of the whole string. A value of "BOTTOM" causes the bottom of the text extent rectangle to pass through the text position. At the start of program execution, the values for text justification shall be "LEFT" and "BASE."

Figure 2 illustrates the text attributes associated with "JUSTIFY."

Figure 2. Text Attributes Associated with JUSTIFY



Execution of an ask-statement with the keywords TEXT JUSTIFY shall assign the current values of the horizontal and vertical components of text justification to the first and second string-variables, respectively. The values returned shall be all upper-case-letters.

13.2.5 Exceptions

A color index in a set-statement with the keywords COLOR MIX, POINT COLOR, LINE COLOR, TEXT COLOR, or AREA COLOR is less than zero or greater than the maximum color index for the implementation (11085, nonfatal: use the implementation default).

The value of the numeric-expression in a set-statement with the keywords LINE STYLE is less than or equal to zero or greater than the maximum style available (11062, nonfatal: use the value one).

The value of the numeric-expression in a set-statement with the keywords POINT STYLE is less than or equal to zero or greater than the maximum style available (11056, nonfatal: use the value three).

The value of the numeric-expression in a set-statement with the keywords TEXT HEIGHT is less than or equal to zero (11073, nonfatal: use the current value).

The value of one of the string-expressions in a set-statement with the keywords TEXT JUSTIFY is not one of those listed in the semantics above (4102, nonfatal: use the current values).

The value of a numeric-expression used to set a color proportion in a set-statement with the keywords COLOR MIX is less than zero or greater than one (11088, nonfatal: use the current values).

13.2.6 Remarks

It is recommended that implementations make the value returned by ASK MAX COLOR the same as the number of colors (not counting background color) available for simultaneous display, not the total number of different colors available on the device.

An implementation may predefine the color mix associated with any or all color index values.

If possible, the width of characters, in problem coordinates, should be kept in constant proportion to their height, as determined by TEXT HEIGHT.

On a monochrome device, it is recommended that the intensity be set to .30*RED + .59*GREEN + .11*BLUE.

The CLEAR statement corresponds to the GKS function CLEAR WORKSTATION (#0,CONDITIONALLY). SET LINE STYLE and SET POINT STYLE corresponds to the GKS functions SET LINETYPE and SET MARKER TYPE, respectively. SET LINE COLOR, SET POINT COLOR, SET TEXT COLOR, and SET AREA COLOR correspond to the GKS functions SET POLYLINE COLOUR INDEX, SET POLYMARKER COLOUR INDEX, SET TEXT COLOUR INDEX, and SET FILL AREA COLOUR INDEX, respectively. SET TEXT HEIGHT corresponds to the GKS function SET CHARACTER HEIGHT. SET TEXT ANGLE X may be translated to the GKS function SET CHARACTER UP VECTOR (Cos(X+PI/2),Sin(X+PI/2)). SET JUSTIFY corresponds to the GKS function SET TEXT ALIGNMENT. SET COLOR MIX corresponds to the GKS function SET COLOR REPRESENTATION.

The following ask-objects correspond to various parameters of the GKS function INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES: LINE STYLE is linetype, POINT STYLE is marker type, LINE COLOR is polyline colour index, POINT COLOR is polymarker colour index,

TEXT COLOR is text colour index, and AREA COLOR is fill area colour index.

The following ask-objects may be derived from various parameters of the GKS function INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUE: TEXT HEIGHT is character height, TEXT ANGLE is computed as ANGLE(DX,DY)-PI/2 where DX and DY are from character up value. TEXT JUSTIFY is text alignment.

ASK COLOR MIX corresponds to the index and color parameters of the GKS function INQUIRE COLOUR REPRESENTATION with workstation zero and REALIZED returned values. ASK MAX LINE STYLE corresponds to the number of available line types parameter of the GKS function INQUIRE POLYLINE FACILITIES. ASK MAX POINT STYLE corresponds to the number of available marker types parameter of the GKS function INQUIRE POLYMARKER FACILITIES. ASK MAX COLOR returns a value that is one less than the value from the "number of color table entries" parameter of the GKS function INQUIRE LIST OF COLOR INDICES for GKS implementations in which the parameter indicates the total number of colors available for simultaneous display.

13.3 Graphic Output

13.3.1 General Description

The statements described in this section are used to generate various kinds of graphic output. The user may cause points, line segments, or filled-in areas to be drawn on the screen. There is a facility for including text within the drawing. Finally, there is a facility for generating an array of colored cells. The effect of the graphic output statements depends on the current values of the various set-objects described in 13.1 and 13.2. Additional graphic output capability is provided in 13.5.

13.3.2 Syntax

```
 1. imperative-statement        > graphic-output-statement
 2. graphic-output-statement    = geometric-statement /
                                  array-geometric-statement /
                                  graphic-text-statement /
                                  array-cells-statement
 3. geometric-statement         > graphic-verb geometric-object
                                  colon point-list
 4. graphic-verb                > GRAPH
 5. geometric-object            = POINTS / LINES / AREA
 6. point-list                  = coordinate-pair (semicolon
                                  coordinate-pair)*
 7. coordinate-pair             = numeric-expression comma
                                  numeric-expression
 8. array-geometric-statement   = MAT graphic-verb
                                  geometric-object (comma
                                  size-select)? colon
                                  array-point-list
 9. size-select                 = LIMIT index
10. array-point-list            = numeric-array (comma
                                  numeric-array)?
11. graphic-text-statement      = graphic-verb TEXT initial-point
                                  (comma USING image colon
                                  expression-list /
                                  colon string-expression)
12. initial-point               = comma AT coordinate-pair
13. array-cells-statement       = MAT graphic-verb CELLS comma IN
                                  point-pair colon numeric-array
14. point-pair                  = coordinate-pair semicolon
                                  coordinate-pair
```

When an array-point-list has two numeric-arrays, they both shall be one-dimensional; when it has only one, the array shall be two-dimensional.

A geometric-statement with LINES as the geometric-object shall contain at least two coordinate-pairs in its point-list; a geometric-statement with AREA as the geometric-object shall contain at least three coordinate-pairs in its point-list.

The numeric-array in the array-cells-statement shall be two dimensional.

13.3.3 Examples

```
 3. GRAPH LINES: 3,4; 5,6; 66.66,77.77
 8. MAT GRAPH POINTS: XY_PTS
    MAT GRAPH AREA, LIMIT 7: X,Y
11. GRAPH TEXT, AT XP,YP: "here is the label: " & TEXT$
    GRAPH TEXT, AT 0,Y_VALUE, USING "##.##^^^^": Y_VALUE
13. MAT GRAPH CELLS, IN p_1_x, p_1_y; p_2_x, p_2_y: color_array
```

13.3.4 Semantics

13.3.4.1 The Graphic-Output-Statement. Graphic-output-statements are the means by which the user generates all graphic output. The geometric-statement and array-geometric-statement are used to draw a series of marked points, a contiguous set of line segments, or a filled polygon area. The graphic-text-statement produces alphanumeric labels. The array-cells-statement generates a set of colored rectangular cells within an encompassing boundary rectangle.

The output generated by print-statements, input-statements, and trace-statements shall not affect the output generated by graphic-output-statements.

13.3.4.2 The Geometric-Statement and Array-Geometric-Statement. The geometric-statement and array-geometric-statement both make use of a sequence of points specified in problem coordinates. For the geometric-statement, that sequence is determined by the coordinate-pairs in the point-list, the first coordinate-pair designating the first point and so on through the end of the point-list. For the array-geometric-statement, the sequence is determined by the size-select and array-point-list, as follows. If a size-select is present, the index shall be evaluated by rounding to an integer to obtain the number of points. If no size-select is present, then the number of points is the length of the vectors, or the number of elements in the

first dimension of the array, depending on which is used in the array-point-list. If the array-point-list consists of two vectors, then the x-coordinates of the points shall be taken from the first vector and the y-coordinates from the second vector. If the array-point-list consists of an matrix of size two in the second dimension, then the x-coordinates shall be taken from the array column with the lower subscript, and the y-coordinates from the column with the higher subscript. The sequence of points shall always be taken in order beginning with the first row of the array, or first elements of the vectors. The sequence shall be terminated by the end of the matrix or vectors if no size-select is specified, or when the number of points specified in the size-select has been reached.

If the geometric-object is POINTS, then a point marker of the style and color indicated by the current value of POINT STYLE and POINT COLOR shall be drawn at each point in the sequence. If the geometric-object is LINES, then a line segment shall be drawn connecting each successive pair of points in the sequence, the first to the second, the second to the third, and so on. Thus, the number of line segments shall be one fewer than the number of points in the sequence. The style and color of the segments are determined by the current value of LINE STYLE and LINE COLOR. If the geometric-object is AREA, then a filled polygon is drawn whose edges consist of the sequence of line segments as described above for LINES. If the first and last points in the sequence are not coincident, then the line segment joining them completes the outline. The color of the interior and edge is determined by the current value of AREA COLOR. The interior of the polygon is defined as the set of all points (pixels) such that any line segment beginning at that point and extended indefinitely in any direction will cross the polygon boundary an odd number of times. The fill pattern shall be solid on devices where this is possible.

The effect of an array-point-list containing an numeric-array whose size in the second dimension is greater than two is implementation-defined.

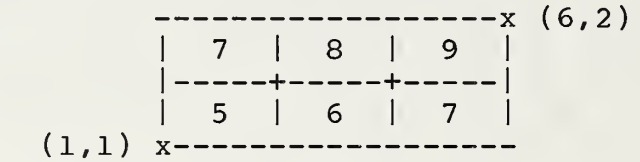
13.3.4.3 The Graphic-Text-Statement. The graphic-text-statement draws a label consisting of the string of characters generated by its string-expression, or by its image and expression-list. The characters generated in the latter case shall be as described in 10.4. The characters used for labels shall have an implementation-defined size, style, and orientation. The effect of clipping on characters that lie partly in and partly out of the viewport on the screen is implementation-defined.

The manner in which a label is displayed shall by governed by the current value of TEXT HEIGHT, TEXT JUSTIFY, and TEXT ANGLE.

13.3.4.4 The Array-Cells-Statement. The array-cells-statement is used to fill a rectangular area. The diagonally opposite corners of this rectangle are given in problem coordinates by the two points in the point-pair. The numeric-array's current size determines the number of cells in each direction, the first dimension corresponding to the horizontal, the second dimension to the vertical. Each cell shall be of equal size, with a width of ABS(first x-coordinate - second x-coordinate) / size of first dimension, and a height of ABS(first y-coordinate - second y-coordinate) / size of second dimension. Each cell shall be filled with the color whose index is the rounded value of the corresponding element of the numeric-array. The cell located at the first point specified corresponds to the array element with the lowest subscripts in both dimensions. The cell located at the second point corresponds to the array element with the highest subscript in both dimensions. Thus, for example, if the numeric-array A has values:

```
A(1,1) = 5,   A(2,1) = 6,   A(3,1) = 7,
A(1,2) = 7,   A(2,2) = 8,   A(3,2) = 9,
```

and the points are (1,1) and (6,2), then the resulting rectangle will be:



where the index designates the color of the cell.

If the output device is not capable of producing pixel-oriented output, it shall simulate cell output. The minimal simulation required is to draw the transformed boundaries of the cell rectangle.

13.3.5 Exceptions

The number of items in two vectors in an array-point-list are not the same and no size-select has been specified (6401, fatal).

Only one numeric-array has been specified in an array-point-list and the size of its second dimension is less than 2 (6401, fatal).

The value of the index of a size-select is less than 1 or greater than the number of points available in the array-point-list (6402, fatal).

A graphic-output-statement with LINES as the geometric-object specifies fewer than two points (11100, fatal).

A graphic-output-statement with AREA as the geometric-object specifies fewer than three points (11100, fatal).

A color index specified by the numeric-array in an array-cells-statement is not available (11085, nonfatal: use the implementation-defined default).

13.3.6 Remarks

The graphic-text-statement is designed to give easy access to a device's hardware-generated character set. For example, the character orientation and direction of rotated text is not defined.

Text is described with respect to problem coordinates and may become distorted when the aspect ratio of the window and viewport differ.

If a device is unable to fill a polygon, it is recommended that the outline of the polygon be drawn and the interior be hashed or shaded in a manner corresponding to the current color number.

It is recommended that the result of filling an area consisting solely of colinear points be a line segment through those points, that filling or drawing a line through a set of coincident points result in a dot being drawn, and that a zero width or zero height rectangular area for cell array output result in a line of cells affected.

GRAPH POINTS and MAT GRAPH POINTS correspond to the GKS function POLYMARKER. GRAPH LINES and MAT GRAPH LINES correspond to the GKS function POLYLINE. GRAPH AREA and MAT GRAPH AREA correspond to the GKS function FILL AREA. GRAPH TEXT is an extension of the GKS function TEXT in that it allows formatting of text with USING. MAT GRAPH CELLS corresponds to the GKS function CELL ARRAY.

No exception is mandated for a two-dimensional numeric-array in an array-point-list when the size of its second dimension is greater than 2. This is to allow future extensions of this standard to specify three-dimensional graphical output. Implementations may choose to give exception 6401 in this case if they support only those graphics capabilities defined in this standard.

The GKS function UPDATE WORKSTATION #0: "SUPPRESS" is implicitly called whenever output is directed to the non-graphics device associated with channel zero by print-statements, input-statements, nonfatal exceptions, or trace-statements.

13.4 Graphic Input

13.4.1 General Description

Input may be supplied to a program from one or more devices associated with a graphics work station. A point or an array of points may be obtained from such devices as cross-hairs on a display or a mouse. An integer value may be entered through a choice of buttons or menu items. A continuous value may be entered from a dial or similar device. Several of each type of input device may be attached to the workstation.

It is also possible to determine the number of picture elements (pixels) in a given rectangle of a raster display. The color of these pixels may be determined.

13.4.2 Syntax

```
 1. imperative-statement    > graphic-input-statement
 2. graphic-input-statement > locate-statement /
                              array-locate-statement
 3. locate-statement        = LOCATE (point-select colon
                              coordinate-variables /
                              value-select colon
                              numeric-variable)
 4. array-locate-statement  = MAT LOCATE  point-select colon
                              array-locate-object
 5. point-select            = POINT device-select? initial-point?
 6. coordinate-variables    = numeric-variable comma
                              numeric-variable
 7. value-select            = CHOICE device-select?
                              start-value? /
                              VALUE device-select? range-select?
                              start-value?
 8. array-locate-object     = redim-numeric-array (comma
                              redim-numeric-array)? /
                              numeric-variable-vector comma
                              numeric-variable-vector /
                              numeric-variable-matrix
 9. redim-numeric-array     = numeric-array redim?
10. numeric-variable-vector = numeric-array left-parenthesis
                               question-mark right-parenthesis
11. numeric-variable-matrix = numeric-array left-parenthesis
                               question-mark comma
                               right-parenthesis
12. range-select            = comma RANGE numeric-expression TO
                              numeric-expression
```

```
13. device-select          = left-parenthesis index
                             right-parenthesis
14. start-value            = comma AT numeric-expression
15. ask-object             > MAX device-type DEVICE
                             numeric-variable /
                             PIXEL SIZE left-parenthesis
                             point-pair right-parenthesis
                             numeric-variable comma
                             numeric-variable /
                             PIXEL ARRAY point-location
                             numeric-array (comma
                             string-variable)? /
                             PIXEL VALUE point-location
                             numeric-variable
16. point-location         = left-parenthesis coordinate-pair
                             right-parenthesis
17. device-type            = POINT / MULTIPOINT / CHOICE /
                             VALUE
```

When a single redim-numeric-array is used as an array-locate-object it shall be two-dimensional. When a pair of redim-numeric-arrays is used, as an array-locate-object they shall both be one-dimensional.

The numeric-array in an ask-object with the keywords PIXEL ARRAY shall be two-dimensional.

A numeric-array used in a numeric-variable-vector shall be one-dimensional. A numeric-array used in a numeric-variable-matrix shall be two-dimensional.

13.4.3 Examples

```
3.  LOCATE POINT : X_IN, Y_IN
    LOCATE POINT, AT 4.7,5.2: wide, long
    LOCATE VALUE(3), range -7 TO 7: Amps
    Locate choice: button

4.  MAT LOCATE POINT: XVALS(?), YVALS(?)
    Mat locate point(indev), at 0,0: Points(2,10)
    MAT LOCATE POINT: SKETCH(?,)
    mat locate point: times, temps

15. MAX CHOICE DEVICE Last_button_set
    MAX MULTIPOINT DEVICE NSTROKES
    PIXEL SIZE(.5,2.5;0,0) xvals, yvals
    Pixel array (.5,2.5) Pix_vals
    pixel value(timel,weight) curpoint
```

```
! Find the colors of pixels in unit square at (0,0)
Option Base 1
Dim Pix(30,30)  ! expected maximum of 30x30 pixels
Ask pixel size(0,0; 1,1) one, two
If one*two <= 900 AND one > 0 and two > 0 then
   Mat pix = zer(one,two)  ! redim to right size
   Ask pixel array(0,1) pix
End if
```

13.4.4 Semantics

Locate-statements and array-locate-statements are similar to input-statements and array-input-statements. Instead of supplying values in terms of characters, the operator supplies input by positioning a cursor to a point or points, pressing a button, rotating a dial, or similar operation. Users are notified of a need for graphic input by an implementation-defined means. If a workstation does not have at least one device for any device-type then device number one of that type shall be simulated through a keyboard or other appropriate means.

If an initial-point is specified in a graphic-input-statement, an indicator (e.g., cursor or tracking cross) shall be positioned at that point in problem coordinates, before the user supplies input to a graphic-input-statement. If no initial-point is specified, the position of any indicators is implementation-defined.

Execution of a locate-statement with the keyword POINT shall assign the problem coordinates of a single point to the coordinate-variables with the x-coordinate going to the first variable and the y-coordinate going to the second-variable. The position returned is always within the current window (i.e., the problem coordinates established by the most recently executed SET WINDOW or the default coordinates, if no SET WINDOW has been executed). If an attempt is made to supply values for a point outside the current window, device window, or device viewport, the effect is implementation-defined (see 13.4.6).

Execution of an array-locate-statement shall assign the problem coordinates of one or more points to the array-locate-object. Whenever the array-locate-object consists of two one-dimensional arrays, the value of the the horizontal coordinate is always assigned to elements of the first array, and the value of the vertical coordinate to the second. When the array-locate-object is a two-dimensional array, the horizontal coordinate is assigned to the lower numbered column and the vertical coordinate to the higher numbered column. Dynamic redimensioning of redim-

numeric-arrays shall be done prior to input in accordance with the rules for dynamic redimensioning during array input in 10.5.4. If the locate-object is a pair of one-dimensional arrays, they shall both be of the same length after any redimensioning. The number of points input is exactly the size of the vectors. If the locate-object is a single redim-numeric-array, its second dimension shall be of size 2 after any redimensioning. The number of points input is the size of the first dimension. If the array-locate-object is a numeric-variable-matrix or a pair of numeric-variable-vectors, a variable number of points are input. Points are input until the operator signals by a device-dependent means that all have been sent. Assignment to a pair of vectors begins at the current lower bound for each vector. When all points have been assigned, the numeric-variable-vectors shall be redimensioned dynamically by setting each upper bound to the subscript of the element receiving the last point. Assignment to a numeric-variable-matrix proceeds in a similar manner, begining with the current lower bound of the second subscript and assigning the x-coordinates to the first column and the y-coordinates to the second. When all points have been assigned, the first dimension is redimensioned by setting the upper bound of the first subscript to the subscript of the element receiving the last point and the upper bound of the second subscript to one more than the current lower bound of the second subscript. If an attempt is made to supply values for a point outside the current window, device window, or device viewport, the effect is implementation-defined (see 13.4.6).

Execution of a locate-statement with the keyword CHOICE shall assign a positive integer value to the numeric-variable. The value assigned shall correspond to the response from a choice device. The maximum value shall correspond to the number of operator's choices available. If the device allows the operator to signify that no choice was made and the operator does so, a value of zero shall be returned. The meaning of a start-value is device-dependent and may not be possible for all devices. When possible, the input device shall position an indicator at the value designated in the start-value.

Execution of a locate-statement with the keyword VALUE shall assign a real value to the numeric-variable. The value assigned shall be the scaled measure of a valuator device when the operator indicates it is set. Scaling is done such that the lowest physical value from the device maps to the value of the first numeric-expression in the range-select and the highest physical value from the device maps to the value of the second numeric-expression in the range-select. Intermediate values are

scaled proportionately. If no range-select is present, the range from the most recently executed locate-statement with the keyword VALUE and the same device shall be used. If no such values have been set, the default range shall be [0,1]. The meaning of a start-value is device-dependent and may not be possible on some devices. Where possible, the input device shall position an indicator at the value designated in the start-value.

If a device-select is specified in a locate-statement or array-locate-statement, the index shall be rounded to an integer N and shall be used to select the Nth device of the appropriate class attached to the workstation. If no device-select is present, a value of one shall be used. Each class of graphic input device (POINT, MULTIPOINT, VALUE, CHOICE) has its own set of device-select values, starting with one for the first device of that class. An exception shall occur if a selected device does not exist.

Execution of an ask-statement with the keywords MAX POINT DEVICE shall return the number of devices available for use in a locate-statement with the keyword POINT.

Execution of an ask-statement with the keywords MAX MULTIPOINT DEVICE shall return the number of devices available for use in an array-locate-statement.

Execution of an ask-statement with the keywords MAX CHOICE DEVICE shall return the number of devices available for use in a locate-statement with the keyword CHOICE.

Execution of an ask-statement with the keywords MAX VALUE DEVICE shall return the number of devices available for use in a locate-statement with the keyword VALUE.

A point-pair specifies, in problem coordinates, diagonally opposite corners of a rectangle, in any order. Execution of an ask-statement with the keywords PIXEL SIZE shall assign to the numeric-variables the number of pixels in the horizontal and vertical directions, in that order, of pixels whose positions lie within the rectangle. If the number of pixels cannot be determined (e.g., a non-raster device), zeroes shall be assigned to the numeric-variables. A value of zero shall always be returned for a status-clause associated with PIXEL SIZE.

Execution of an ask-statement with the keywords PIXEL ARRAY shall assign to the numeric-array the color indices of the rectangle of pixels whose upper-left corner is given in problem coordinates by the point-location. The number of of pixels in

the horizontal direction in the rectangle is the number of elements in the first dimension of the numeric-array. The number of pixels in the vertical direction in the rectangle is the number of elements in the second dimension of the numeric-array. If the color index of a pixel cannot be determined, a value of -1 is returned for that cell. A value of -1 is appropriate for points outside the display surface or for all points of a non-raster device. If a string-variable is present, a value of "PRESENT" in upper-case-letters shall be assigned if any undeterminable points are present, otherwise a value of "ABSENT" shall be returned.

Execution of an ask-statement with the keywords PIXEL VALUE shall assign to the numeric-variable the color index of the pixel located at the problem coordinates specified in point-location. If the index cannot be determined, a value of -1 shall be assigned.

When an ask-statement with the keywords PIXEL ARRAY or PIXEL VALUE contains a status-clause, a value of 11040 shall be returned in the numeric-variable of the status-clause if no pixel readback capability is present.

13.4.5 Exceptions

The number of elements required for a redimensioned array exceeds the number of elements reserved by the array's original dimensions (5001, fatal).

The length of two vectors in an array-locate-object are not equal and they are not numeric-variable-vectors (6401, fatal)

The graphic input device specified through a device-select is not present on the workstation (11140, fatal).

Only one numeric-array has been specified in an array-locate-object and the size of its second dimension is less than 2 (6401, fatal).

A start-value in a locate-statement with the keyword CHOICE is less than zero or greater than the maximum choice number for the device (11152, fatal).

A start-value in a locate-statement with the keyword VALUE is outside the range currently defined for the device (11152, fatal).

An initial-point is outside the device viewport after the normalization and device transformations are applied (11152, fatal).

13.4.6 Remarks

The meaning of an numeric-array used as an array-locate-object whose second dimension is of size greater than two is implementation-defined.

It is recommended that implementations report an attempt to perform graphic input with POINT outside the current window, device window, or device viewport as an exception (8106, nonfatal: request that graphic input be re-supplied) to permit interception by exception-handlers.

LOCATE POINT corresponds to a subset of the GKS function REQUEST LOCATOR. An initial-point specified with LOCATE POINT corresponds to a subset of the GKS function INITIALISE LOCATOR. MAT LOCATE POINT corresponds to a subset of the GKS function REQUEST STROKE. An initial-point specified with MAT LOCATE POINT corresponds to a subset of the GKS function INITIALISE STROKE. When transforming a point from DC-space to problem coordinates, GKS transform number 1 is the default.

LOCATE VALUE corresponds to a subset of the GKS function REQUEST VALUATOR. The start-value and range-select specified with LOCATE VALUE corresponds to a subset of the GKS function INITIALISE VALUATOR.

LOCATE CHOICE corresponds to a subset of the GKS function REQUEST CHOICE. A statt-value specified with LOCATE CHOICE corresponds to the GKS function INITIALISE CHOICE.

The GKS functions SET LOCATOR MODE, SET STROKE MODE, SET VALUATOR MODE, and SET CHOICE MODE are implicitly set to "REQUEST."

ASK MAX POINT DEVICE, ASK MAX MULTIPOINT DEVICE, ASK MAX CHOICE DEVICE, and ASK MAX VALUE DEVICE correspond to the locator, stroke, choice, and valuator parameters of the GKS function INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES.

ASK PIXEL SIZE, ASK PIXEL ARRAY, and ASK PIXEL VALUE correspond to the GKS functions INQUIRE PIXEL ARRAY DIMENSIONS, INQUIRE PIXEL ARRAY, and INQUIRE PIXEL, respectively.

13.5 Graphic Pictures and Moving Point Output

13.5.1 General Description

This subsection provides graphics capabilities that are not defined in the GKS standard.

Graphic procedures for drawing pictures can be defined in a manner similar to that for subprograms. Such picture-defs can be invoked and, in addition, a modeling transformation (such as scaling, shifting, or rotating) can be applied to graphic input and output within such procedures.

A capability for drawing lines connecting points specified in more than one statement is also provided. It is also possible to apply the inverse of this modeling transform upon input.

13.5.2 Syntax

```
 1. imperative-statement    > draw-statement /
                              exit-picture-statement /
                              transform-assignment
 2. draw-statement          = DRAW picture-invocation (WITH
                              transform)?
 3. picture-invocation      = picture-name
                              procedure-argument-list?
 4. picture-name            = routine-identifier
 5. transform               = transform-term (asterisk
                              transform-term)*
 6. transform-term          = transform-function
                              function-arg-list /
                              numeric-array / current-transform
 7. transform-function      = ROTATE / SHEAR / SHIFT / SCALE
 8. exit-picture-statement  = EXIT PICTURE
 9. transform-assignment    = MAT numeric-array equals-sign
                               transform
10. graphic-verb            > PLOT
11. geometric-statement     > PLOT LINES / (PLOT LINES colon
                              point-list semicolon)
12. picture-def             = internal-picture-def /
                              external-picture-def
13. procedure               > external-picture-def
14. internal-proc-def       > internal-picture-def
15. internal-picture-def    = internal-picture-line
                              block* end-picture-line
16. internal-picture-line   = line-number picture-statement tail
17. picture-statement       = PICTURE picture-name
                              procedure-parm-list?
```

```
18. end-picture-line        = line-number END PICTURE tail
19. external-picture-def    = external-picture-line
                              unit-block* end-picture-line
20. external-picture-line   = line-number EXTERNAL
                              picture-statement tail
21. type-declaration        > internal-picture-type /
                              external-picture-type
22. internal-picture-type   = PICTURE picture-name-list
23. picture-name-list       = picture-name (comma picture-name)*
24. external-picture-type   = EXTERNAL PICTURE picture-name-list
25. line                    > end-picture-line /
                              external-picture-line /
                              internal-picture-line
26. current-transform       = TRANSFORM
27. graphic-input-statement > GET point-select colon
                               coordinate-variables /
                               MAT GET point-select colon
                               array-locate-object
28. numeric-array-value     > TRANSFORM
```

No line-number in a control-transfer outside an internal-picture-def shall refer to a line in an internal-picture-def other than an internal-picture-line, nor shall a line-number in a control-transfer inside an internal-picture-def refer either to a line outside that internal-picture-def or to the associated internal-picture-line.

A line-number in a control-transfer inside an external-picture-def shall not refer to the associated external-picture-line.

If a picture-name is defined by an external-picture-def, it shall not be defined more than once in the program. If a picture-name is defined by an internal-picture-def, it shall not be defined more than once in the containing program-unit.

Within a program-unit, no more than one picture (internal or external) of a given name shall be declared or defined.

If a picture-name is defined by an external-picture-def, then a declare-statement with external-picture-type containing that picture-name shall occur in a lower-numbered line than the first reference to that picture-name in a draw-statement in the same program-unit.

If a picture-name is defined by an internal-picture-def, then either the internal-picture-def or a declare-statement with internal-picture-type containing that picture-name shall occur in

a lower-numbered line than the first reference to that picture-name in a draw-statement in the same program-unit.

Self-recursive pictures need not declare themselves; that is, if a picture-def contains a reference to itself in a draw-statement, that reference does not require a type-declaration containing that picture-name in a lower-numbered line.

An exit-picture statement shall occur only within a picture-def.

The number and type of procedure-arguments in a draw-statement shall agree with the number and type of procedure-parameters in the corresponding picture-def. These rules are the same as for subprograms (see 9.2).

An actual-array shall have the same number of dimensions as the corresponding formal-array. The number of dimensions in a formal-array is one more than the number of commas in the formal-array.

Whenever a numeric argument is passed to a corresponding numeric parameter in a different program-unit, the ARITHMETIC options in effect for the two program-units shall agree.

A given procedure-parameter shall occur only once in a procedure-parm-list. Procedure-parameters shall not be explicitly declared or dimensioned within the internal- or external-picture-def.

The channel-number #0 shall not be used as a procedure-parameter.

A picture-name appearing in an internal-picture-type shall be defined elsewhere in the same program-unit by an internal-picture-def.

A picture-name appearing in an external-picture-type shall be defined elsewhere in the program by an external-picture-def.

All numeric-arrays in a transform or transform-assignment shall be two-dimensional.

The number and type of the arguments for transform-functions shall agree with the specifications given below in the semantics.

13.5.3 Examples

```
 2. DRAW CIRCLE
    DRAW CIRCLE WITH SHIFT(2,0) * SCALE(.4)
    DRAW HOUSE ("SPLIT-LEVEL", 80000) WITH SHEAR(.1)

 8. EXIT PICTURE

 9. MAT SPIN_SHRINK = ROTATE(180)*SCALE(.1)
    MAT Where_am_I = Transform

11. Plot lines: x,sin(x);

19. 100 EXTERNAL PICTURE CIRCLE
    150    LET P = 2*3.14159265 / 50
    200    FOR I = 0 TO 50
    250        LET IP = I*P
    300        PLOT LINES: COS(IP),SIN(IP);
    400    NEXT I
    500 END PICTURE
```

13.5.4 Semantics

Execution of a draw-statement shall cause the picture-def with the same picture-name to be executed. Pictures are invoked in a manner analogous to that for subprograms. The actual parameters are passed according to the same rules and both may be invoked recursively. Scope rules for the routine-identifier and variables within picture-defs are also the same as for subprograms and depend on whether the picture-def is internal or external. The rules for channel numbers in subprograms apply to picture-defs as well. There are two important differences, however. First, set-statements that redefine coordinate systems (i.e., those with the keywords WINDOW, VIEWPORT, DEVICE WINDOW, or DEVICE VIEWPORT) or that affect clipping (i.e., those with the keyword CLIP) shall not be executed during the execution of a picture-def. And second, all graphic output generated in a graphic-output-statement whose graphic-verb is PLOT shall be the same as that generated when the graphic-verb is GRAPH with points transformed as follows by the transformations specified in the picture-invocation for the picture-def. When such a statement is executed and no picture-def is in effect, then the graphic output is the same as described in 13.3. Graphic output generated by graphic-output-statements with the graphic-verb GRAPH is unaffected by the transformations that may be in effect for a picture.

For the purpose of effecting transformations, the problem coordinates (x,y) of a point in two dimensions shall be represented by the "homogeneous" coordinates (cx,cy,0,c), where c is any nonzero number. Coordinates shall be transformed by post-multiplying this row vector by a 4x4 matrix, which is the value of the transform in the draw-statement, to obtain a new vector of homogeneous coordinates. Particular transformations may be effected through the use of transform-functions, which have one or two numeric-expressions as arguments and which produce 4x4 matrices as their values. The effects and values of these transform-functions, as well as the number of arguments required by each, shall be as follows.

Function	Transformations
SHIFT(A,B)	Translates (x,y) to (x+A,y+B). Returns: 1 0 0 0 0 1 0 0 0 0 1 0 A B 0 1
SCALE(A,B)	Scales (x,y) to (Ax,By). Returns: A 0 0 0 0 B 0 0 0 0 1 0 0 0 0 1
SCALE(A)	Scales (x,y) to (Ax,Ay). Returns: A 0 0 0 0 A 0 0 0 0 A 0 0 0 0 1
ROTATE(A)	Rotates (x,y) A degrees or radians counterclockwise about the origin of the problem Returns: cos(A) sin(A) 0 0 -sin(A) cos(A) 0 0 0 0 1 0 0 0 0 1 .
SHEAR(A)	Shears vertical lines in the (x,y)-plane to lean by A degrees or radians by mapping (x,y) to (x+y*tan(A),y). Returns: 1 0 0 0 tan(A) 1 0 0 0 0 1 0 0 0 0 1

The ROTATE and SHEAR functions work with arguments in degrees or radians according to the setting for OPTION ANGLE for the program unit (cf. 5.6).

The parameterless current-transform function TRANSFORM shall return the current accumulated transform as a 4x4 matrix. In a program-unit that has not been called directly or indirectly through a draw-statement, TRANSFORM shall return the identity matrix.

Numeric-arrays used as transform-terms shall be 4x4 matrices. The transformation effected by such a transform is thereby that obtained by performing, from left to right, the transformations associated with each of the transform-terms.

A picture-def may invoke other procedures; it may also invoke itself, either directly or indirectly through another procedure.

If, in the course of executing one picture-def, a second picture-def is invoked with a transform, then two (or possibly more) transforms shall be applied to the points generated by that second picture-def: first the transform specified in the draw-statement invoking the second picture-def, and then any transforms being applied to the first picture-def.

The end-picture-line serves both to mark the textual end of a picture-def and, when executed, to terminate execution of the picture-def. The exit-picture-statement, when executed, shall terminate execution of the picture-def. When execution of the picture-def terminates, execution shall continue at the line following the draw-statement that initiated execution of the picture-def.

Execution of a stop-statement in a picture-def shall terminate execution of the entire program.

When a transform-assignment is executed, the 4x4 matrix that is the value of the transform shall be assigned to the numeric-array to the left of the equals-sign, dynamically redimensioning that numeric-array if necessary in accordance with the rules for redimensioning stated in 7.2.

13.5.4.1 Transformed Input. Execution of a graphic-input-statement with the keywords GET POINT or MAT GET POINT differs from that of a graphic-input-statement with the keywords LOCATE POINT or MAT LOCATE POINT in only one respect. Points input when GET is used are transformed through the inverse of the current-

transform in addition to the normalization and device transformations. If the current-transform cannot be inverted (i.e., if it is a singular matrix), the effect of a GET statement is implementation-defined.

13.5.4.2 Moving Point Output. Execution of a geometric-statement with the keywords PLOT LINES differs from that of a geometric-statement with the keywords GRAPH LINES in three ways. First, points are potentially transformed through picture invocation as described above. Second, there is no exception for fewer than two points in a point-list. Third, the actual lines drawn may depend on the previously executed geometric-statement with the keywords PLOT LINES. The points in point-lists are determined in the same manner as described in 13.3. Output may be described in terms of movement of a beam on a screen. In reality, the beam and screen may be a pen and a sheet of paper on a plotter or some other combination of physical devices. The beam is either on or off. If it is moved when it is on, it draws a line on the screen connecting two points. If the beam is moved when it is turned off, there is no effect on the screen. Execution of a geometric-statement with the keywords PLOT LINES causes the beam to move from its current position to each of the points in the point-list (if any) and, at the end of each move, the beam is turned on. At the end of the execution of such statements, the beam shall remain on if and only if the list of points ends with a semicolon. The beam shall be turned off and its position shall be undefined in the following cases: (1) following execution of a geometric-statement with the keywords PLOT LINES that did not have a point-list or whose point-list did not end with a semicolon, (2) before executing any graphic-output statement not containing the keywords PLOT LINES, (3) at the start of program execution, (4) before executing any graphic-input-statement, (5) before executing any statement that affects the NDC space transformation (i.e., set-statements with the keywords WINDOW, VIEWPORT, DEVICE WINDOW, or DEVICE VIEWPORT), (6) upon invocation of or exit from picture-defs.

Thus, for example, the following picture-def will cause an approximation of a circle to be drawn, assuming reasonable scaling.

```
110 External picture circle
120     Let P = 2*PI/100
130     For I = 0 to 100
140         Plot lines cos(I*P), sin(I*P);
150     Next I
160 End picture
```

13.5.5 Exceptions

A set-statement that redefines coordinates or affects clipping is executed during execution of a picture-def (11004, fatal).

A transform used in a draw-statement is not a 4x4 matrix (6201, fatal).

The original dimensions of the numeric-array being assigned a value in a transform-assignment do not permit that numeric-array to be redimensioned to a 4x4 matrix (5002, fatal).

13.5.6 Remarks

Transforms are represented as 4x4 matrices to permit extensions to three dimensions.

All exceptions defined in 13.3.5 apply to graphic-output statements defined in this subsection, except for the exception caused by fewer than 2 points in a plot-list with a graphic-object LINES.

All exceptions defined in 13.4.5 apply to graphic-input-statements defined in this subsection.

The only effect of a PLOT LINES statement with no point-list or semicolon is to turn the beam off.

Remarks about the following topics in 9.1.6 apply analogously to pictures:

(1) Program-units with different COLLATE options.

(2) Functions that are defined or declared, but not referenced.

(3) Functions that are defined before they are declared.

(4) The requirement that external, but not internal, functions always be declared (rather than defined).

(5) Internal functions with the same name in different program-units.

Remarks on aliases in 9.2.6 also apply to picture-defs.

14. Real-Time

The real-time module of this standard is intended for use in applications involving control, automation, and monitoring. It enables a program to be divided into a number of concurrent single-thread activities which cooperate to achieve the overall objective of the application.

Facilities are provided to schedule execution of concurrent activities so that they may respond to both internally and externally generated events. Communication between concurrent activities is possible either through the use of shared data or by the transmission of messages.

A process object is a part of the external environment of a real-time-program. Typical process objects are measurement or control points in a plant interface. Communication between a concurrent activity and a process object is accomplished by input and output operations accessing the process object through a process port.

14.1 Real-Time Programs

14.1.1 General Description

A real-time-program is composed of real-time declarations (cf. 14.2) that describe a process environment, one or more parallel-sections, and some number of procedures that may be invoked by these parallel-sections. Each parallel-section is named and is delimited by the keywords PARACT (PARallel ACTivity) and END PARACT. Parallel-sections constitute separate program-units and serve to define concurrent activities.

Execution of a parallel-section is enabled by a scheduling-statement (cf. 14.3) and starts at the first line of the parallel-section.

14.1.2 Syntax

```
1. program                > program-name-line?
                            real-time-program
2. real-time-program      = real-time-declarations
                            parallel-section parallel-section*
                            procedure-part*
3. program-unit           > parallel-section
4. parallel-section       = remark-line* paract-line
                            block* end-paract-line
```

```
 5. line                    > paract-line / end-paract-line
 6. paract-line             = line-number paract-statement tail
 7. paract-statement        = PARACT routine-identifier
                              (URGENCY urgency)?
 8. urgency                 = integer
 9. end-paract-line         = line-number end-paract-statement
                              tail
10. end-paract-statement    = END PARACT
11. statement               > real-time-statement
12. real-time-statement     = parstop-statement /
                              scheduling-statement /
                              process-io-statement /
                              data-io-statement /
                              message-io-statement /
                              exit-seize-statement
13. parstop-statement       = PARSTOP
14. block                   > real-time-block
15. real-time-block         = select-port-block / seize-block
```

A given routine-identifier shall not occur in more than one paract-statement in a real-time-program.

All program-units in a real-time-program shall use the same arithmetic option.

Real-time-statements and real-time-blocks shall occur only in parallel-sections.

14.1.3 Examples

```
4. 320 PARACT Rigl
   330     WAIT TIME 17*60*60
   340     PRINT "Time to go home"
   350 END PARACT
```

14.1.4 Semantics

Execution of a parallel-section in a real-time-program shall constitute a concurrent activity. At any point in the execution of a real-time-program, a concurrent activity may be in one of the following states:

(1) In progress, that is, in the initial state of the concurrent activity defined by the lexically first parallel-section, or in the state of a concurrent activity following execution of a start-statement naming that activity

(2) Stopped, that is, not yet in progress, or formerly in progress but subsequently terminated by execution of a parstop-statement, an end-paract-statement, or a statement generating a fatal exception that is not inhibited by the action of an exception-handler

(3) Waiting, that is, formerly in progress but suspended by execution of a wait-statement or message-io-statement, until the occurrence of a specified event, the passing of a specified length of time, the arrival of a specified time of day, or the exchange of messages.

Several concurrent activities may be in progress at any given time. Initially, the only concurrent activity in progress shall be that defined by the lexically first parallel-section in the real-time-program; other concurrent activities shall be placed in progress only by the execution of start-statements (cf. 14.3). If there is a parameter-list on the program-name-line with a real-time-program, its parameters are in the scope of the lexically first parallel-section.

The urgency of a parallel-section shall indicate to the scheduler the relative importance of the concurrent activity. A lower value shall indicate a greater importance. The precise interpretation of the urgency shall be implementation-defined.

Each time a parallel-section is placed in progress by a start-statement, the values of all variables shall be implementation-defined according to the usual policy for variable initialization.

Lines in a parallel-section shall be executed in sequential order, starting at the first line of the parallel-section, until:

(1) Some other action is dictated by the execution of a line
(2) An exception occurs
(3) A stop-statement, chain-statement, parstop-statement, or an end-paract-statement is executed

Execution of a parstop-statement or of an end-paract-statement shall close all files local to the parallel-section, and terminate execution of the concurrent activity in which it occurs, causing that activity to stop until placed in progress again by another execution of a start-statement. Execution of a stop-statement or a chain-statement shall terminate execution of the entire real-time-program. The occurrence of a fatal exception that is not handled by an exception-handler shall stop the concurrent activity in which it occurs. If an implementation

is such that an exception can occur when execution reaches a real-time declaration, then that exception shall terminate execution of the entire real-time program.

Each parallel-section is a distinct entity in that identifiers used to name variables, arrays, internal functions, and exception handlers shall be local to the parallel-section, (i.e., they shall name different objects in different parallel-sections). Identifiers used to name supplied functions, parallel-sections, procedures defined as program-units, process-ports, process-port-arrays, events, structures, message ports, and data ports shall be global to the entire real-time-program (i.e., they shall name the same object wherever they occur).

14.1.5 Exceptions

None.

14.1.6 Remarks

Execution of a concurrent activity may be interrupted at implementation-defined times in order to execute other concurrent activities that are in progress.

Possible interpretations of the urgency of a parallel-section might be the priority of that section or a deadline for execution of the parallel-section.

14.2 Real-Time Declarations

14.2.1 General Description

Concurrent activities communicate with the external environment through process ports. Process port declarations define the names of these ports and the attributes of process-objects in a real-time system attached to these ports. process-objects may be either active or passive. Passive process-objects are typically measurement and control points in a plant interface, such as temperature sensors or stepping motor controllers (cf. 14.4). Active process-objects, or process-events, are typically sources of program interrupts, such as timers and alarms (cf. 14.3).

Data ports provide a means of accessing data whose scope is wider than an individual concurrent activity. A data port declaration defines the name of a data port and the structure of the data accessible through it (cf. 14.5).

Message ports provide a means of transferring data between two concurrent activities; the data transferred does not exist outside the two activities. A message port declaration defines the name of a message-port and the structure of the data transferred through it. A message is sent when the same message-port-name is used in two concurrent activities: in a send-statement in one and a receive-statement in the other (cf. 14.6).

Data structure declarations provide a means of specifying the structure of data transferred through process, data, and message ports. They enable a language processor to check the validity of statements sending and receiving data through a port, and they specify indivisible units of shared data.

14.2.2 Syntax

```
1. real-time-declarations    = (remark-line / rt-declare-line /
                                process-array-dec)*
2. line                      > rt-declare-line /
                               process-declare-line /
                               process-element-line /
                               end-process-line
3. rt-declare-line           = line-number rt-declare-statement
                               tail
4. rt-declare-statement      = DECLARE (data-structure-dec /
                               process-port-dec /
                               data-port-dec /
```

```
                                  message-port-dec)
 5. data-structure-dec        = STRUCTURE structure-name colon
                                  repeat-count? type
                                  (comma repeat-count? type)*
 6. structure-name            = letter identifier-character*
 7. repeat-count              = integer OF
 8. type                      = (NUMERIC fixed-point-type? /
                                  STRING) bounds?
 9. process-port-dec          = PROCESS
                                  (process-clause / event-clause)
                                  access-information?
10. process-clause            = io-qualifier process-port-name
                                  OF structure-name
11. io-qualifier              = INPUT / OUTPUT / OUTIN
12. process-port-name         = letter identifier-character*
13. process-array-dec         = process-declare-line
                                  process-element-line*
                                  end-process-line
14. process-declare-line      = line-number
                                  process-declare-statement tail
15. process-declare-statement = DECLARE PROCESS io-qualifier
                                  process-port-array bounds
                                  OF structure-name
16. process-port-array        = letter identifier-character*
17. process-element-line      = line-number
                                  process-element-statement tail
18. process-element-statement = process-port-array
                                  left-parenthesis signed-integer
                                  (comma signed-integer)?
                                  right-parenthesis colon
                                  access-information
19. end-process-line          = line-number END PROCESS tail
20. event-clause              = EVENT event-name
21. event-name                = letter identifier-character*
22. access-information        = string-constant
23. data-port-dec             = SHARED data-port-name
                                  bounds? OF structure-name
24. data-port-name            = letter identifier-character*
25. message-port-dec          = MESSAGE message-port-name
                                  OF structure-name
26. message-port-name         = letter identifier-character*
27. identifier                > real-time-identifier
28. real-time-identifier      = structure-name / event-name /
                                  process-port-name /
                                  process-port-array /
                                  data-port-name /
                                  message-port-name
```

Any structure-name appearing in a process-clause, process-declare-statement, data-port-dec, or message-port-dec shall be defined in a data-structure-dec in a lower-numbered line.

A given real-time-identifier may name a structure, an event, a process-port, a process-port-array, a data-port, or a message-port but not more than one of these.

The value of the integer in a repeat-count shall be greater than zero.

Within a process-array-dec, there shall be exactly one process-element-statement for each element of the array, as determined by the bounds in the process-declare-statement. Process elements shall be described in ascending row major order.

The name and dimensionality of the process-port-array in the process-declare-statement and its process-element-statements shall match.

14.2.3 Examples

```
 5. STRUCTURE OPR: STRING, 2 OF NUMERIC, NUMERIC(10)
    STRUCTURE A1: 2 OF NUMERIC
    STRUCTURE B1: NUMERIC
 9. PROCESS INPUT WEIGHT OF A1 "ADCCHAN 3"
    PROCESS OUTIN PANEL OF OPR "Q, 177640"
    PROCESS INPUT A "BCD 4"
    PROCESS OUTPUT Z1 OF B1
    PROCESS EVENT FULL "INT 36"
13. 100 DECLARE PROCESS INPUT xyz(2, -1 TO 1) OF A1
    110     xyz(1, -1): "wb slot 10"
    120     xyz(1,  0): "wb slot 11"
    130     xyz(1,  1): "wb slot 12"
    140     xyz(2, -1): "wb slot 20"
    150     xyz(2,  0): "wb slot 21"
    160     xyz(2,  1): "wb slot 22"
    170 END PROCESS
23. SHARED FLIGHT(10) OF OPR
    SHARED D OF B1
25. MESSAGE LINK OF OPR
```

14.2.4 Semantics

The scope of structure-names, process-port-names, process-port-arrays, data-port-names, message-port-names, and event-names shall be all the parallel-sections in a real-time program.

A data-structure-dec shall declare the name of a data structure for use in process-port-decs, process-array-decs, data-port-decs, and message-port-decs. A data structure is an abstract structure (i.e., one without any storage allocated to it) consisting of an ordered list of types that may be either numeric or string, scalar or array. A repeat-count shall specify the number of occurrences of the type that follows it. An omitted repeat-count implies one occurrence of the type.

A process-port-dec shall define the name of a process port and the attributes of a process-object in a real-time system attached to that port. Each process-array-dec shall declare an array of process-ports. The dimensionality and maximum and minimum values for subscripts of the array shall be determined by the bounds in the usual way (see 7.1). A process-element-statement shall declare the access-information peculiar to each element of the array.

The presence of a process-clause shall indicate that the process-object attached to that process-port is passive. Processes declared in process-array-decs are always passive. The io-qualifier shall indicate the permitted directions of data transfer through the port: INPUT shall indicate that the process-object provides input only, OUTPUT that it accepts output only, and OUTIN that it supports both input and output.

When a structure-name is specified in a process-clause or process-array-dec, then the named structure is associated with that process-port, or with each element of the process-port-array. In the absence of a structure-name, the default data structure shall be a single numeric.

The presence of an event-clause in a process-port-dec shall declare the named process-object to be active (i.e., to be a process-event). When connected, a process-event shall be capable of generating events that return concurrent activities waiting for them to the state of being in progress (cf. 14.3).

Access-information for a process-port specifies a particular process-object attached to that port and the format of its data. Access information for an active process-object typically specifies the source of a hardware interrupt signaling the occurrence of an event associated with that object together with information about how to control the interrupt. The interpretation of the access-information or its absence shall be implementation-defined.

A data-port-dec shall define the name of a data-port and the structure of the data accessible through it. If bounds are specified in a data-port-dec, then they shall define an array of instances of the given structure. The array so defined shall be either one-, two-, or three-dimensional according to whether one, two, or three bounds-ranges are specified in the bounds. If no bounds appear, a single instance of the given structure shall be defined. Shared data shall be accessible by all concurrent activities (cf. 14.5).

A message-port-dec shall define the name of a message port and the structure of the data transferred through it.

14.2.5 Exception

The attributes of a process object do not match its declaration (12201, fatal).

14.2.6 Remarks

process-port-arrays can only be arrays of passive process-objects (i.e., arrays of process-events are not permitted).

The format information in the access-information for a process-port may allow the implementation to perform automatic data transformation, such as scaling or conversion between a character-coded external representation of a number in a process-object and a floating-point internal representation. An implementation may also allow names of routines in the access-information so that special devices can be handled by standard mechanisms invoked automatically each time a process-port is accessed. These routines could, for example, handle access via a multiplexer with a long switching time or handle special Gray code devices.

Fixed-point-type is defined in the optional fixed decimal module (cf. Section 16), and may not be available if that module is not implemented.

14.3 Scheduling

14.3.1 General Description

The scheduling requirements for concurrent activities are specified by execution of start-statements and wait-statements. A start-statement places a concurrent activity in progress. The actual execution of concurrent activities that are in progress is scheduled by the implementation according to the urgency of these activities. A wait-statement can be used to suspend execution of a concurrent activity for a specified period of time, until a given time, or until a specified event occurs. Events may be generated externally by connected process objects or internally by execution of signal-statements.

Connect-statements and disconnect-statements referring to events are used to enable and disable specific event signals from the external hardware.

14.3.2 Syntax

```
 1. scheduling-statement    = start-statement / wait-statement /
                              signal-statement /
                              connect-statement /
                              disconnect-statement
 2. start-statement         = START routine-identifier
 3. wait-statement          = WAIT (wait-time / wait-interval /
                              wait-event )
 4. wait-time               = TIME time-expression
 5. time-expression         = numeric-time-expression /
                              string-time-expression
 6. string-time-expression  = string-expression
 7. wait-interval           = DELAY numeric-time-expression
 8. wait-event              = EVENT event-name
                              timeout-expression?
 9. signal-statement        = SIGNAL event-name
10. connect-statement       = CONNECT EVENT event-list
11. event-list              = event-name (comma event-name)*
12. disconnect-statement    = DISCONNECT EVENT event-list
```

An event-name that does not occur in a process-port-dec shall not occur in a connect-statement or a disconnect-statement.

An event-name that occurs in a process-port-dec shall not occur in a signal-statement.

A routine-identifier that occurs in a start-statement shall also occur in some paract-line in the program. An event-name

that occurs in a wait-statement shall occur in a signal-statement or shall be declared as an event in a process-port-dec.

14.3.3 Examples

```
 2. START FILL
 3. WAIT DELAY 1.5*60*60
    WAIT TIME "09:15:00"
    WAIT EVENT READY TIMEOUT 4
    WAIT TIME A$
 9. SIGNAL READY
10. CONNECT EVENT FULL
12. DISCONNECT EVENT FULL, TOOFUL
```

14.3.4 Semantics

Execution of a start-statement shall place in progress the concurrent activity defined by the named parallel-section. Execution of a wait-statement shall cause the concurrent activity in which it occurs to be suspended for a specified period of time, until a specified time, or until a specified event occurs.

The value of a numeric-time-expression shall be interpreted as specifying a number of seconds. If the value of the expression is not an integer, then the accuracy of the time expression is dependent on the resolution of the timer. The value of a string-time-expression shall conform to the format, range of values, and interpretation of the TIME$ function (cf. 6.4).

If a wait-statement specifies a wait-interval, then the concurrent activity shall be suspended for the specified length of time, being placed in progress again when that time has elapsed. If a wait-statement specifies a wait-time with a numeric-time-expression, then the concurrent activity shall be suspended until the specified number of seconds have elapsed since the previous midnight, at which time it shall be placed in progress again. If the number of seconds since the previous midnight has already elapsed, then the concurrent activity shall wait until that time the following day. If a wait-statement specifies a wait-time with a string-time-expression, then the concurrent activity shall be suspended until the specified time of day, at which time it shall be placed in progress again. If the specified time of day has already passed, then the concurrent activity shall wait until that time the following day.

If a wait-statement specifies a wait-event, then the concurrent activity shall be suspended until that event occurs,

at which time it shall be placed in progress again (cf. 14.2 and 14.4). If a timeout expression is specified in a wait-event, then an exception shall occur if the specified event has not occurred within the specified length of time.

Execution of a signal-statement shall cause the specified event to occur. Following execution of a signal-statement, the concurrent activity continues to be in progress.

Execution of a connect-statement shall cause the specified events to be connected. A connected process object can cause events to occur.

Execution of a disconnect-statement shall cause the specified events to be not connected, and shall cause any previous occurrence of that event not acted upon by a wait-statement to have not occurred. A process object that is not connected shall not cause events to occur.

An event that has occurred shall place in progress again a concurrent activity waiting for the event. If no concurrent activity is waiting for the event, then the first concurrent activity subsequently to execute a wait-statement naming that event shall remain in progress. In either case, the event shall then be deemed to have not occurred.

If more than one concurrent activity is waiting for the same event, then which one of those activities shall be placed in progress upon occurrence of that event shall be implementation-defined. Only one concurrent activity shall be placed in progress upon each occurrence of an event.

If a new event is caused by a signal-statement before a previous occurrence of the same event has been acted upon by a wait-statement, then that signal-statement shall cause an exception. The events shall then be deemed to have not occurred.

If a new event is generated by a connected process-object before a previous event generated by that object has been acted upon by a wait-statement, then the next wait-statement to be executed that names that event shall cause an exception. The events shall then be deemed to have not occurred.

At the initiation of execution of a real-time-program, all events shall have not occurred, and all process-events shall be not connected.

14.3.5 Exceptions

A start-statement is executed that specifies a concurrent activity that is not stopped (12001, fatal).

A signal-statement is executed that specifies an event that has already occurred, but that has not yet caused a waiting concurrent activity to be placed in progress again (12002, fatal).

The value of a numeric-expression used as a time-expression exceeds 86400, the number of seconds in a day, or is less than zero (12004, fatal).

The value of a string-expression used as a time-expression does not conform to the format of the TIME$ function (12005, fatal).

The event specified in a wait-statement does not occur within the period of time specified in a timeout-clause (12101, fatal).

A wait-statement is issued for an event generated by a connected process object that has occurred more than once since the last such event was consumed by a wait-statement (12003, fatal).

A connect-statement is executed that specifies an event that is already connected (12006, fatal).

A disconnect-statement is executed that specifies an event that is already disconnected (12007, fatal).

14.3.6 Remarks

When the system clock requires adjustment, such as for seasonal time changes or to correct for errors, problems can arise with wait-statements specifying wait-times. In particular, if the clock is moved back, any activities that were released from a wait-time during the previous occurrence of that time should not be put in progress again until the following day. Similarly, if the clock is advanced, activities waiting for a time that is "passed over" should be put in progress as if that time had occurred.

14.4 Process Input and Output

14.4.1 General Description

In-statements and out-statements are used to move data over communication paths between passive process-objects and a real-time-program. An in-statement permits external values to be transferred to program variables, and an out-statement permits the transfer of values to external process-objects.

14.4.2 Syntax

```
1. process-io-statement   = in-statement / out-statement
2. in-statement           = IN FROM (process-port-name /
                              process-port-array subscript-part)
                              TO in-structure timeout-expression?
3. in-structure           = in-structure-element
                              (comma in-structure-element)*
4. in-structure-element   = variable / array-name
5. out-statement          = OUT TO (process-port-name /
                              process-port-array subscript-part)
                              FROM out-structure
                              timeout-expression?
6. out-structure          = out-structure-element
                              (comma out-structure-element)*
7. out-structure-element  = expression / array-name
```

Any process-port-name or process-port-array occurring in an in-statement or out-statement shall be declared in a process-port-dec, or process-array-dec.

The number of subscripts in a subscript-part in an in-statement or an out-statement shall equal the number of bounds-ranges in the bounds of the corresponding process-array-dec.

The number and type of elements within an in-structure or out-structure shall conform element by element, with the data-structure-dec for the structure specified in the declaration for the corresponding process port. That is,

(1) The number of in-structure- or out-structure-elements shall be the same as the number (counting repeats) of types in the corresponding data-structure-dec

(2) The in-structure- or out-structure-elements in the in- or out-structure shall be associated with the corresponding types in the data-structure-dec (i.e., the first with the first, the

second with the second, and so on), and the types shall correspond as follows:

data-structure-dec type	in-structure-element	out-structure-element
NUMERIC	simple-numeric-variable	numeric-expression
STRING	simple-string-variable	string-expression
NUMERIC (bounds)	numeric-array	numeric-array
STRING (bounds)	string-array	string-array

The dimensionality of arrays specified as in-structure- or out-structure-elements shall match that of the corresponding data-structure-dec type. If a data-structure-dec type contains a fixed-point-type, then the corresponding in-structure- or out-structure-elements shall have that same type (cf. 15.1).

A process-port-name or process-port-array declared as OUTPUT shall not appear in an in-statement, nor shall one declared as INPUT appear in an out-statement.

14.4.3 Examples

```
2. IN FROM WEIGHT TO X, Y
   IN FROM PANEL TO A$, B, C, Fvect
   IN FROM RIG1(NEXT) TO ALPHA TIMEOUT 2.5
5. OUT TO Z1 FROM B*C+X
   OUT TO PANEL FROM A$&B$, JIM, FRED, Cvect
```

14.4.4 Semantics

Execution of an in-statement shall cause values to be obtained from the specified process-port and to be assigned to the corresponding variables and arrays in the in-structure. The order of assignment to variables and evaluation of subscripts and substring-qualifiers is the same as in the input-statement.

If a numeric value causes an underflow, then its value shall be replaced by zero.

Execution of an in-statement shall be regarded as complete only when all values have been assigned to the variables and arrays in the in-structure or when a fatal exception occurs, such as one caused by incorrect data or a hardware failure, or the number of seconds specified by the timeout-expression has expired.

Execution of an out-statement shall cause the expressions in the out-structure to be evaluated and their values, together with the values of all elements in the specified formal-arrays, to be transmitted to the specified process-port.

Execution of an out-statement shall be regarded as complete only when all values from the out-structure have been accepted by the process environment or when a fatal exception occurs, such as one caused by incorrect data or a hardware failure, or the number of seconds specified by the timeout-expression has expired.

The occurrence of an array in an in-structure or an out-structure shall cause the contents of the entire array with that name to be input or output, in row major order.

14.4.5 Exceptions

The assignment of a value to a numeric-variable or numeric-array in an in-structure causes a numeric overflow (1201, fatal).

The assignment of a value to a string-variable or string-array in an in-structure causes a string overflow (1203, fatal).

The current sizes of the dimensions of an array used in an in-structure or an out-structure do not conform to the data-structure-dec for the structure specified in the declaration for the indicated process port (6301, fatal).

Execution of an in-statement or an out-statement has not been completed before the timeout given by the timeout-expression has expired (12102, fatal).

A subscript for a process port is not within the range specified by the process-array-dec (2001, fatal).

14.4.6 Remarks

Implementation-defined exception conditions may exist. These are mainly concerned with the characteristics of particular process-objects.

14.5 Shared Data

14.5.1 General Description

Get-statements and put-statements are used to transmit data between concurrent activities and collections of shared data. The data are transmitted through data ports.

14.5.2 Syntax

1. data-io-statement = put-statement / get-statement
2. put-statement = PUT TO data-port-name subscript-part? FROM out-structure timeout-expression?
3. get-statement = GET FROM data-port-name subscript-part? TO in-structure timeout-expression?

Any data-port-name occurring in a put-statement or get-statement shall be declared in a data-port-dec. A subscript-part shall follow the data-port-name if and only if a bounds occurs in the data-port-dec for that data-port-name; in that case, the number of subscripts in the subscript-part shall equal the number of dimensions specified by the bounds.

The number and types of elements within an in-structure or out-structure shall conform to the data-structure-dec for the structure named in the data-port-dec for that data-port-name, as specified in 14.4.

14.5.3 Examples

```
2. PUT TO FLIGHT(N+1) FROM I$, N, 2, P
3. GET FROM D TO E
```

14.5.4 Semantics

Execution of a put-statement shall cause the expressions in the out-structure to be evaluated and their values, together with the values of all elements in the specified arrays, to be transmitted to the appropriate collection of the shared data.

Execution of a get-statement shall cause the variables and arrays in the in-structure to be assigned values from the appropriate collection of shared data. Subscripts and substring-qualifiers in an in-structure shall be evaluated after values have been assigned to the variables and arrays preceding them (i.e., to the left of them) in the in-structure.

If a get-statement references a data element that has not been initialized, the result is implementation-defined.

Execution of a put-statement or a get-statement shall be regarded as complete when all values have been transmitted, or when a fatal exception has occurred. No other concurrent activity shall access the specified collection of shared data until execution of a get-statement or put-statement is complete.

Data-io-statements interact with seize-blocks (cf. 14.8) as follows. If the data-io-statement is not contained in a seize-block, it behaves as if it were contained within a seize-block consisting of (1) a seize-line with a single seize-item, namely, the data-port-name of the data-io-statement and with the timeout-expression (if any) of the data-io-statement, (2) the line with the data-io-statement, but without the timeout-expression (if any), and (3) an end-seize-line. If the data-io-statement is contained in an explicit seize-block, no additional constraints are imposed.

14.5.5 Exceptions

The assignment of a value to a numeric-variable or numeric-array in an in-structure causes a numeric overflow (1202, fatal).

The assignment of a value to a string-variable or string-array in an in-structure causes a string overflow (1204, fatal).

The current sizes of the dimensions of an array used in an in-structure or an out-structure do not conform to the data-structure-dec for the structure specified in the declaration for the indicated data port (6301, fatal).

A subscript for a data-port is not within the range specified by the data-port-dec (2001, fatal).

14.5.6 Remarks

None.

14.6 Message Passing

14.6.1 General Description

Send-statements and receive-statements are used to transmit data between concurrent activities. The data are conveyed over message paths that connect a message output port in a send-statement in one concurrent activity to a message input port in a receive-statement in another.

A message path is established at run time implicitly by the use of the same message-port-name in two concurrent activities, in a send-statement in one and in a receive-statement in the other.

14.6.2 Syntax

```
 1. message-io-statement    = send-statement / receive-statement
 2. send-statement          = SEND TO message-port-name
                              FROM out-structure
                              timeout-expression?
 3. receive-statement       = RECEIVE FROM message-port-name TO
                              in-structure timeout-expression?
 4. select-port-block       = select-port-line remark-line*
                              case-port-block case-port-block*
                              case-timeout-block? end-select-line
 5. select-port-line        = line-number select-port-statement
                              tail
 6. select-port-statement   = SELECT ON PORT
 7. case-port-block         = case-port-line block*
 8. case-port-line          = line-number case-port-statement
                              tail
 9. case-port-statement     = CASE (SEND / RECEIVE) MESSAGE
                              message-port-name /
                              CASE EVENT event-name
10. case-timeout-block      = case-timeout-line block*
11. case-timeout-line       = line-number case-timeout-statement
                              tail
12. case-timeout-statement  = CASE TIMEOUT
                              numeric-time-expression
```

Any message-port-name occurring in a send-statement, receive-statement, or select-port-block shall be declared in a message-port-dec.

The number and types of elements in the out-structure of a send-statement and in the in-structure of a receive-statement shall conform to the data-structure-dec for the structure named

in the message-port-dec for that message-port-name, as specified in 14.4.

A parallel-section shall not contain both a send-statement and a receive-statement specifying the same message-port-name.

A case-port-block whose case-port-line specifies SEND MESSAGE shall contain a send-statement for the corresponding message-port-name. A case-port-block whose case-port-line specifies RECEIVE MESSAGE shall contain a receive-statement for the corresponding message-port-name.

A given message-port-name or event-name shall not occur more than once in the case-port-statements of a given select-port-block.

No line number in a control-transfer outside a select-port-block, case-port-block, or case-timeout-block shall refer to a line inside that select-port-block, case-port-block, or case-timeout-block, respectively, other than to the first line of a select-port-block.

An event-name occuring in a case-port-statement shall be declared in a process-port-dec.

14.6.3 Examples

```
2. SEND TO LINK FROM "FIRST", X/2, 17.35, RESULTS
3. RECEIVE FROM LINK TO A$, P(1), P(2), I TIMEOUT 30
4. 100 SELECT ON PORT
   110 CASE EVENT E
   120      PRINT "E occurred at "; time$
   130      START E_processor
   140 CASE SEND MESSAGE X
   150      SEND TO X FROM a, b, c
   160 CASE SEND MESSAGE Y
   170      SEND TO Y FROM d$, e$
   180 CASE RECEIVE MESSAGE Z
   190      PRINT "Someone just sent me a Z"
   200      RECEIVE FROM Z TO f, g$
   210 CASE TIMEOUT 600
   220      PRINT "Ten-minute timeout."
   230 END SELECT
```

14.6.4 Semantics

A message port in one concurrent activity shall be connected to a message port in another concurrent activity by the execution of a send-statement or a receive-statement in the one concurrent activity using the given message-port-name and the subsequent execution in the other concurrent activity of a receive-statement or a send-statement using the same message-port-name.

Execution of a send-statement or a receive-statement shall not be complete until the specified message port has been connected as a result of executing a corresponding receive-statement or send-statement in another concurrent activity, or an exception occurs. Until complete, a process is waiting as described in 14.1.

When such a connection has been made, the expressions in the out-structure in the send-statement shall be evaluated, and their values, together with the values of all arrays in the out-structure, shall be assigned to the corresponding variables and arrays in the in-structure in the corresponding receive-statement.

Subscripts and substring-qualifiers in an in-structure shall be evaluated after values have been assigned to the variables and arrays preceding them (i.e., to the left of them) in the in-structure. Subsequent to this assignment, the message-ports of the two communicating activities are disconnected and the processes return to the state of being in progress.

If a timeout is specified in a send-statement or a receive-statement, then an exception shall occur if no connection is made within the specified length of time.

If a send-statement times out, then its message is no longer available for a receive-statement.

If a send-statement is executed and more than one other concurrent-activity is waiting to receive a message through a message port with the same name, then which one of those activities receives the message shall be implementation-defined. Only one of the activities receives the message; the others continue to wait.

If a receive-statement is executed and more than one other concurrent-activity is waiting to send a message through a message port with the same name, then which one of those activities sends the message shall be implementation-defined.

Only one of the activities sends the message; the others continue to wait.

Execution of a select-port-block shall cause the concurrent activity to be suspended until either (1) a message-io-statement is executed, or has been executed but not connected, that names a message-port-name in one of the case-port-statements and for which the direction of message transmission is opposite to that specified by the case-port-statement (i.e., a send-statement is needed to activate CASE RECEIVE MESSAGE and a receive-statement to activate CASE SEND MESSAGE), (2) a signal-statement is executed, or has been executed but not caused a concurrent activity to be put in progress, that names an event-name in one of the case-port-statements, (3) the name of one of the event-names in one of the case-port-statements corresponds to an event that occurs or that has occurred but not caused a concurrent activity to be put in progress, or (4) the number of seconds specified in the case-timeout-statement has elapsed.

If the wait on the select-port-block is released as a result of a message-io-statement, then the case-port-block with the corresponding message-port-name and appropriate transmission direction is executed. The message, if any, associated with the above message-io-statement shall only be transferred when the corresponding message-io-statement is executed in the case-port-block.

If the wait on the select-port-block is released as a result of an event occurring, then the case-port-block with the corresponding event-name is executed. The execution of the case-port-block shall cause the event to have not occurred.

If more than one case-port-block may be executed, then which one of them shall be executed shall be implementation-defined.

If there is a case-timeout-block present and no execution has taken place before the number of seconds specified in the case-timeout-statement has elapsed since the select-port-block was entered, then the rest of the case-timeout-block shall be executed.

If control reaches the end of a case-port-block or case-timeout-block, then control shall be transferred to the line following the corresponding end-select-line.

14.6.5 Exceptions

The assignment of a value to a numeric variable or a numeric array in an in-structure causes a numeric overflow (1202, fatal).

The assignment of a value to a string variable or a string array in an in-structure causes a string overflow (1204, fatal).

The current sizes of the dimensions of an array used in an in-structure in a receive-statement or an out-structure in a send-statement do not conform to the data-structure-dec for the structure specified in the declaration for the indicated message port (6301, fatal).

Execution of a send-statement or receive-statement has not been completed before the time specified in a timeout has expired (12103, fatal).

14.6.6 Remarks

It is essential that the underlying system shall select case-port-blocks in such a way as to ensure the activity on a given message port or of a specific event does not inhibit response to other message ports or events.

14.7 Bit Patterns and Operations

14.7.1 General Description

Bit patterns are a common means of coding information in process control systems. Within a program, they are represented by strings of characters. Operations on bit patterns may be performed by the string operations of concatenation and substring extraction.

Functions are provided for conversion between strings and numeric values.

14.7.2 Syntax

1. string-supplied-function > BSTR dollar-sign
2. numeric-supplied-function > BVAL

14.7.3 Examples

None.

14.7.4 Semantics

The values of the supplied functions, as well as the number and types of their arguments, shall be as described below. B$ represents a string expression, R and V represent indices, and R shall take on only the values 2, 8, or 16.

Function	Value
BVAL(B$, R)	The non-negative integer whose string representation is given by the string B$. R is the radix of the string representation of the value, e.g.: BVAL("101", 2) = 5 BVAL("2F", 16) = 47
BSTR$(V, R)	The string representation of the value of V, using radix R. Unless a fatal exception occurs, BSTR$ shall always return at least one character. In particular, the value of BSTR$ when V is zero is "0", but otherwise no leading zeros are returned, e.g.: BSTR$(3.14, 2) = "11" BSTR$(15, 8) = "17"

The permissible characters that may appear in the string B$ depends on the value of R. If R is 2, the valid set consists of the digits 0 and 1. If R is 8, the valid set consists of the digits 0 to 7. If R is 16 the valid set consists of the digits 0 to 9 and the upper-case-letters A to F.

14.7.5 Exceptions

The value of the string argument of BVAL is not a valid representation of a number in radix R (4201, fatal).

The numeric interpretation of the value of the string argument of BVAL cannot be exactly represented within the limits of the precision of numeric variables (4202, fatal).

The numeric interpretation of the value of the string argument of BVAL exceeds the largest number representable (1003, fatal).

The rounded value of the first argument of BSTR$ is negative (4203, fatal).

The rounded value of the second argument of BVAL or BSTR$ is not 2, 8, or 16 (4204, fatal).

14.7.6 Remarks

Typical uses for bit patterns are the manipulation of status registers, or of data from process-objects in which individual bits represent specific objects such as switches or indicators.

14.8 Resource Management

14.8.1 General Description

In concurrent activity systems it is sometimes necessary for the application program to have some control over the allocation of serially reusable resources. An example is a program with more than one concurrent activity that accesses the console to print a message containing several lines. To ensure that coherent messages are printed, an activity shall be able to seize the console for its exclusive use until all the lines of a message have been printed. The seize operation includes an optional timeout so that an activity can be programmed so as to not wait indefinitely for resources to become available.

14.8.2 Syntax

```
1. seize-block              = seize-line block* end-seize-line
2. seize-line               = line-number seize-statement tail
3. seize-statement          = SEIZE seize-list
                              timeout-expression?
4. seize-list               = seize-item (comma seize-item)*
5. seize-item               = SHARED data-port-name /
                              [implementation-defined]
6. end-seize-line           = line-number END SEIZE tail
7. exit-seize-statement     = EXIT SEIZE
8. line                     > seize-line / end-seize-line
```

The same seize-item shall not appear more than once in a seize-list.

No line-number in a control-transfer outside a seize-block shall refer to a line inside that seize-block.

No line-number in a control-transfer inside a seize-block shall refer to a line outside that seize-block.

Seize-blocks shall not be nested. A seize-block shall not contain a seize-line other than at the first line of that seize-block. A seize-block shall not contain an end-seize-line other than at the last line of that seize-block.

An exit-seize-statement shall occur only within a seize-block.

In a seize-block, no reference shall be made to a defined-function or procedure defined outside that seize-block.

If a data-io-statement appears in a seize-block, then the data-port-name of the data-io-statement shall appear as a seize-item in the seize-list of the seize-block.

A data-io-statement appearing in a seize-block shall not contain a timeout-expression.

A seize-block shall not contain a gosub- or on-gosub-statement.

14.8.3 Examples

3. SEIZE SHARED DPORT TIMEOUT 10

14.8.4 Semantics

A seize-statement is either successful in seizing all the resources in the list, in which case the seize-statement is deemed to have been successfully completed and control passes to the line following that seize-statement, or it fails because one or more of the resources is not available, in which case no resources are seized for exclusive use by the activity containing the seize-block. If the seize-statement fails, then the activity is put into the waiting state until all the resources requested become available to it. It then seizes them all simultaneously and control passes to the line following that seize-statement. The timeout-expression is optional; if present, an exception occurs if the seize-statement has not been completed within the time specified.

In a seize-list, the occurrence of a data-port-name following the keyword SHARED refers to that of shared data. Other implementation-defined resource names are permitted in a seize-list.

Execution of an end-seize-line releases all the resources seized for exclusive use by the activity.

Execution of an exit-seize-statement shall cause control to pass to the end-seize-line of the seize-block containing the exit-seize-statement.

If the release of a resource simultaneously enables the seize-lists of several seize-statements waiting for the resource, it is implementation-defined which of the seize-statements succeeds in seizing the resource.

If a fatal exception occurs within a seize-block and is not handled within that seize-block, then control passes to an exception-handler outside the seize-block, or the concurrent activity is stopped in accordance with the rules of 12.1 and 14.1. In either case, the resources of the seize-block are released. If an exception-handler outside the seize-block (but in the same invocation of the same program-unit) issues a RETRY, then control shall return to the seize-line of the seize-block and the seize-statement shall wait again for the resources in its seize-list. If the handler issues a CONTINUE, then control shall be transferred to the line lexically following the end-seize-line.

14.8.5 Exception

Execution of a seize-statement has not resulted in successful completion within the number of seconds of its timeout-expression (12102, fatal).

14.8.6 Remarks

The syntax requirement that seize-blocks shall not be nested, and the semantic requirement that either all resources are seized or none are seized, prevents deadlock.

15. Fixed Decimal Numbers

The fixed decimal numbers module of this standard specifies an option in which the values of all numeric variables behave logically as fixed-point decimal numbers with program-defined precisions. Use of this option also implies that numeric-constants and numeric-expressions generally are represented as fixed-point decimal numbers.

The main intent of this data type is to provide an interface with non-Basic processors, and as a result, the precision and accuracy requirements for numeric-expressions are not specified.

15.1 Fixed Decimal Precision

15.1.1 General Description

An option is provided that allows definition of all numeric variables in a program-unit as having fixed-point decimal numbers as values. The specification of this option defines a default precision for the values of variables. In addition, other precision attributes can be specified for individual variables.

15.1.2 Syntax

```
1. option                > ARITHMETIC FIXED fixed-point-type
2. fixed-point-type      = asterisk fixed-point-size
3. numeric-type          > NUMERIC fixed-point-type?
                           fixed-declaration
                           (comma fixed-declaration)*
4. fixed-declaration     = simple-numeric-variable
                           fixed-point-type? /
                           numeric-array-declaration
                           fixed-point-type?
```

An option-statement with an ARITHMETIC FIXED option, if present at all, shall occur in a lower-numbered line than any numeric-expression, numeric-variable, or any declare-statement with NUMERIC in the same program-unit.

A fixed-declaration, if present at all, shall occur in a lower-numbered line than any reference to the variable or array declared therein.

A fixed-point-size may appear in a declare-statement only if the ARITHMETIC FIXED option has been specified for the program-unit.

Variables and arrays shall not be described more than once, in either a declare-statement, a dimension-statement, or as a function- or procedure-parameter.

Fixed-point-size is defined within the optional enhanced native files module; it shall also be available within the optional fixed-decimal module.

15.1.3 Examples

```
1.  ARITHMETIC FIXED*8.2
3.  NUMERIC*5.2 A, B, C*5.5, D (1 TO 8)*6.6
    NUMERIC     E (1 TO 10, 1961 TO 1981)
```

15.1.4 Semantics

If the ARITHMETIC FIXED option is specified, then the values of numeric constants, variables, and expressions shall behave logically as fixed-point decimal numbers. In the case of variables, this means that the set of values they are capable of assuming are exactly those values that can be expressed with integer-size decimal digits to the left of the decimal point and fraction-size digits to the right, together with the sign. The sign is not counted in the size of the representation. Each implementation shall define a maximum precision, P, that controls the number of decimal digits available for the representation of numeric values. This precision shall not be less than 18.

The semantics for numeric-constants shall be as specified in 5.1, except as follows. Each numeric-constant has a precision attribute defined by the number of significant decimal digits. The first significant digit is either the first nonzero digit, or the digit immediately to the right of the decimal point, whichever is farther left. The last significant digit is the last explicitly written or the digit immediately to the left of the decimal point, whichever is farther right. In the special case of zero, there is at least one significant digit, namely immediately to the left of the decimal point. A numeric-constant written in scaled notation is interpreted as if expressed in the equivalent unscaled notation. For example:

Constant as Written	Significant Digits
12.34	12.34
12.300	12.300
12.300E-4	.0012300
12.300E7	123000000.
00.00E-3	.00000
0.0E3	0.

If the number of significant digits exceeds P, the implementation shall round the value to no fewer than P digits. If the number of significant digits to the left of the decimal point exceeds P, an overflow exception shall result.

Each simple-numeric-variable and numeric-array has a precision attribute defined in terms of the number of digits maintained in the integer part and the fraction part, namely integer-size and fraction-size. The representation of variables and arrays is governed by (in descending order of precedence): (1) the fixed-point-size specified in the fixed-declaration of the variable or array, (2) the fixed-point-size following DECLARE NUMERIC in a declare-statement containing a fixed-declaration for the variable or array, or (3) the default fixed-point-size specified in the ARITHMETIC FIXED option. The precision attribute of a numeric-array applies to each of its elements. The significant digits for a numeric-variable are the same as if the variable were written out with its full precision as a numeric-constant.

The semantics of numeric-expressions and numeric-supplied-functions are as specified in 5.3 and 5.4, except as follows: the precision attribute of the result obtained by evaluating a numeric-supplied-function or numeric-expression is implementation-defined. The accuracy of such evaluation is also implementation-defined.

Assignment of a numeric value to a numeric-variable, whether done by internal assignment (such as with LET), or from an external source (such as with INPUT), proceeds as follows. The integer part and fraction part of the value are moved to the integer part and fraction part of the variable, aligned on the decimal point. If any nonzero digits are truncated on the left of the integer part, an overflow exception results. If any digits are truncated on the right of the fraction part, the resulting value in the variable is rounded to the precision of the variable. If necessary, the value is extended with zeros on the left of the integer part or the right of the fraction part so as to fill all the digit places of the receiving variable.

The semantics for input and output are as specified in Section 10, except as follows: When a numeric-expression is used as a print-item in a print-statement, its value is always printed in unscaled representation, either implicit point or explicit point. The digits printed are exactly the significant digits, as defined above. Note that for numeric-expressions other than variables and constants, significant digits are implementation-defined. Implicit point representation shall be used when there

are no significant digits to the right of the decimal point; otherwise, explicit point representation shall be used.

The semantics for files are as specified in Section 11, with the following additions. For DISPLAY records, the rules given above for input, output, and assignment are used. For INTERNAL records, the values are self-typed, and so input and output takes place as defined above. Thus, the result of WRITE A and READ B, where they access the same value in the file, is the same as LET B = A. Note that it is implementation-defined whether an INTERNAL file accessed with one ARITHMETIC option is accessible with another. The ARITHMETIC FIXED option with different default precisions is not considered to be a different option for the purposes of accessibility. For NATIVE records, assignment of values to and from fields of fixed-point-size takes place in accordance with the usual rules for fixed-point assignment. When a value is moved to a field with a size of E (indicating floating-point), at least the first P significant digits shall be retained exactly. When a value obtained from such a field is assigned to a variable, the field shall be treated as a scaled numeric-constant, as described above. Again, note that it is implementation-defined whether a NATIVE file accessed with one ARITHMETIC option is accessible with another. The ask-attribute DATUM returns the type of the next datum in a file. For numeric data in STREAM INTERNAL files written with the FIXED option, the type returned by the ask-attribute DATUM shall be: "NUMERIC*ii.ff", where ii is the two-digit number for integer-size and ff the two-digit number for fraction-size.

15.1.5 Exceptions

The number of significant digits in the integer part of a numeric-constant exceeds P (1001, fatal).

In an option-statement or a declare-statement, the sum of integer-size and fraction-size in a fixed-point-size exceeds P (1010, fatal).

Upon assignment of a numeric value to a variable, the number of significant digits in the integer part exceeds the variable's integer-size (1011, fatal).

15.1.6 Remarks

It is recommended that the accuracy of transcendental functions such as LOG, COSH, ATN, SIN, and EXP be no less than that specified for OPTION ARITHMETIC DECIMAL. It is recommended that for non-transcendental functions, such as ABS, INT, and MAX,

and for the operations +, -, *, /, and ^, that an intermediate result be maintained as floating-point decimal with P+2 significant digits. This would imply that whenever all the intermediate results of a numeric-expression are exact within P+2 decimal digits and the final result is exact within P decimal digits, then the final result value is exactly correct.

15.2 Fixed Decimal Program Segmentation

15.2.1 General Description

When the fixed decimal option is specified for a program-unit, it applies to all numeric entities in the scope of that unit, including parameters, formal parameters, and (in the case of an external-function-def) the result of function evaluation.

15.2.2 Syntax

```
1. function-parameter          > numeric-fixed-parameter
2. procedure-parameter         > numeric-fixed-parameter
3. numeric-fixed-parameter     = simple-numeric-variable
                                 fixed-point-type /
                                 fixed-formal-array
4. fixed-formal-array          = formal-array fixed-point-type
5. internal-function-line      > line-number FUNCTION
                                 fixed-defined-function
                                 function-parm-list? tail
6. external-function-line      > line-number EXTERNAL FUNCTION
                                 fixed-defined-function
                                 function-parm-list? tail
7. numeric-def-statement       > DEF fixed-defined-function
                                 function-parm-list? equals-sign
                                 numeric-expression
8. defined-function            > fixed-defined-function
9. fixed-defined-function      = numeric-defined-function
                                 fixed-point-type
```

A numeric-variable or actual-array appearing in a call-statement shall have the same integer-size and fraction-size as the corresponding procedure-parameter. For all other numeric function- or procedure-arguments and their corresponding function- or procedure-parameters, it is necessary only that the ARITHMETIC options of their respective program-units agree (i.e., that both be DECIMAL or NATIVE or FIXED).

An option-statement with an ARITHMETIC FIXED option shall occur in a lower-numbered line within the program-unit than any internal-function-def that declares a numeric-defined-function or specifies numeric parameters.

A fixed-point-type may appear in an internal-function-line or numeric-def-statement only if the ARITHMETIC FIXED option has been specified for the program-unit.

A numeric-fixed-parameter may be used only if the ARITHMETIC FIXED option has been specified for the program-unit.

When a fixed-defined-function is declared in a function-type, the fixed-point-sizes specified either explicitly or by default in the declaration and the corresponding definition shall agree.

15.2.3 Examples

```
3. A()*8.2
   B(,)*4.4
5. 123 FUNCTION SUMVECTOR*5.2 (V()*5.2)
6. 234 EXTERNAL FUNCTION ANSWER*1 (A$)
7. DEF AVERAGE*10.3 (X*10.3, Y*10.3) = (X+Y)/2
```

15.2.4 Semantics

When the ARITHMETIC FIXED option is specified for a program-unit, the values of numeric procedure-parameters, numeric function-parameters, and numeric-defined-functions shall be represented and manipulated as fixed-point decimal numbers. If the fixed-point-size of one of these is not explicitly specified in the appropriate procedure- or function-parameter, internal-def-line or internal- or external-function-line, then it is assumed to be the default specified in the ARITHMETIC FIXED option. The fixed-point-size of a formal-array applies to each of its elements.

The evaluation and assignment of the arguments in a function reference to the parameters of the function-def shall proceed as described in 9.1, with the following addition: in the case in which the fixed-point-size of an argument differs from that of the corresponding parameter, the assignment rules given in 15.1.4 shall apply.

The association of the procedure-arguments in a call-statement with the procedure-parameters in the corresponding sub-statement shall proceed as described in 9.2, with the following addition: in the case where the fixed-point-size of an procedure-argument that is a numeric-expression, but not a numeric-variable, differs from that of the corresponding procedure-parameter, the assignment rules given in 15.1.4 shall apply.

15.2.5 Exceptions

In a numeric-fixed-parameter or fixed-defined-function, the sum of the integer-size and fraction-size in a fixed-point-size exceeds P (1010, fatal).

Upon assignment of a numeric value to a numeric-fixed-parameter or fixed-defined-function, the number of significant digits in the integer part exceeds the integer-size of the numeric-fixed-parameter or fixed-defined-function (1011, fatal).

15.2.6 Remarks

None.

16. Editing

The editing module of this standard provides facilities that aid in the construction and modification of programs. The editing module is specifically designed to create a BASIC environment, where a BASIC user may RENUMBER a BASIC program, may LIST it, and may subset it through DELETE and EXTRACT commands. This module is only available during the preparation of a BASIC program. The transition from editing to executing that program is implementation-defined.

16.1 Unsorted Programs

16.1.1 General Description

Editing facilities enable the lines of a program to be entered in an arbitrary order.

16.1.2 Syntax

```
1. unsorted-program  = program-line*
2. program-line      = line-number
                       (character / line-continuation)*
                       end-of-line
```

16.1.3 Examples

```
1. 20 PRINT TWO + TWO
   30 END
   20 PRINT 2 + 2
   15 PRINT "2 + 2 EQUALS"
2. 10 SUB SORT (A(),    & ! List to be sorted
   &            N)       ! Length of list
```

16.1.4 Semantics

The program-lines in an unsorted-program shall be sorted into ascending line-number sequence. In the case of duplicate line-numbers, only the textually last program-line with a given line-number shall be retained.

16.1.5 Exceptions

None.

16.1.6 Remarks

A program-line is a logical line (see 3.2).

16.2 Editing Commands

16.2.1 General Description

Editing commands allow the programmer to manipulate the texts of programs by listing, deleting, or renumbering selected portions of the program.

16.2.2 Syntax

```
 1. edit-command          = delete-command / extract-command /
                            list-command / renumber-command
 2. delete-command        = DELETE segment-list
 3. segment-list          = segment-specifier
                            (comma segment-specifier)*
 4. segment-specifier     = segment-item (TO segment-item)?
 5. segment-item          = line-number / FIRST / LAST
 6. extract-command       = EXTRACT segment-list
 7. list-command          = LIST segment-list?
 8. renumber-command      = RENUMBER segment-specifier?
                            renumber-parameters?
 9. renumber-parameters   = AT initial-number (STEP step-size)? /
                            STEP step-size (AT initial-number)?
10. initial-number        = line-number
11. step-size             = integer
```

Line-numbers in a segment-list shall occur in increasing order. If the keyword FIRST appears, it shall precede all line numbers in a segment-list. If the keyword LAST appears, it shall follow all line numbers in a segment-list.

16.2.3 Examples

```
 2. DELETE 10
 6. EXTRACT 100 TO 200, 300, 500 TO LAST
 7. LIST
    LIST 500 TO 999
 8. RENUMBER
    RENUMBER AT 1000
    RENUMBER AT 1000 STEP 5
    RENUMBER 100 TO 200 STEP 5 AT 100
```

16.2.4 Semantics

A segment-specifier specifies a collection of logical lines in a program: if the segment-specifier is a single segment-item, then the collection shall consist of the indicated line (if the line number does not exist, then the collection is empty); if the

segment-specifier contains two segment-items, then the collection shall consist of all lines whose line-numbers are at least as large as the first, but not larger than the second. A segment-list specifies the collection of lines consisting of all those specified by any segment-specifier in the segment-list. The segment-item FIRST is synonymous with the lowest numbered line in the program and LAST with the highest numbered line in the program.

The delete-command shall cause the specified collection of lines to be deleted from a program.

The extract-command shall cause all lines other than those in the specified collection to be deleted from the program.

The list-command shall cause the text of the specified collection of lines to be displayed; if the segment-list is omitted, the text of the entire program shall be displayed.

The renumber-command shall cause new line-numbers to be assigned to all lines in the specified collection (or in the entire program if the segment-specifier is omitted). The new line-number of the first line in the collection shall be the initial-number (100 if no initial-number is specified); line-numbers for succeeding lines in the collection shall be generated using the specified step-size (or 10 if no step-size is specified). All references throughout the entire program to line-numbers contained in the collection being renumbered shall be changed to conform to the new line-numbers.

The renumber-command shall not be executed, and the text of the program shall be left unchanged, if:

(1) The renumber-parameters are chosen so that renumbered lines would overlay or surround lines not being renumbered

(2) Renumbering would cause a change in the sequence order of lines

(3) Renumbering would resolve an unresolved line-number reference in a control-transfer, formatted-print-list, or template-identifier

(4) Renumbering would create a line-number higher than the host implementation could accept

The implementation shall then inform the user of the reason for the failure.

16.2.5 Exceptions

None.

16.2.6 Remarks

Implementations may issue a warning if a nonexistent line-number is referenced by the segment-specifier for a deletion or extraction.

The commands specified in this standard operate on logical lines only. Implementation-supplied enhancements may provide the capability to edit individual physical lines within a logical line.

It is recommended that the display of the program produced by the list-command show the leading zeros of line-numbers as entered by the user, to facilitate alignment of source code.

Table 8. Standard BASIC Character Set

Ordinal Position	Code	Graphic	ORD Mnemonic	Name
0.	0/0		NUL	Null
1.	0/1		SOH	Start of heading
2.	0/2		STX	Start of text
3.	0/3		ETX	End of text
4.	0/4		EOT	End of transmission
5.	0/5		ENQ	Enquiry
6.	0/6		ACK	Acknowledge
7.	0/7		BEL	Bell
8.	0/8		BS	Backspace
9.	0/9		HT	Horizontal tab
10.	0/10		LF	Line feed
11.	0/11		VT	Vertical tab
12.	0/12		FF	Form feed
13.	0/13		CR	Carriage return
14.	0/14		SO	Shift out
15.	0/15		SI	Shift in
16.	1/0		DLE	Data link escape
17.	1/1		DC1	Device control 1
18.	1/2		DC2	Device control 2
19.	1/3		DC3	Device control 3
20.	1/4		DC4	Device control 4
21.	1/5		NAK	Negative acknowledge
22.	1/6		SYN	Synchronous idle
23.	1/7		ETB	End of trans. block
24.	1/8		CAN	Cancel
25.	1/9		EM	End of medium
26.	1/10		SUB	Substitute
27.	1/11		ESC	Escape
28.	1/12		FS	File separator
29.	1/13		GS	Group separator
30.	1/14		RS	Record separator
31.	1/15		US	Unit separator
32.	2/0		SP	Space
33.	2/1	!		Exclamation point
34.	2/2	"		Quotation mark
35.	2/3	#		Number sign
36.	2/4	$		Dollar sign
37.	2/5	%		Percent sign
38.	2/6	&		Ampersand
39.	2/7	'		Apostrophe
40.	2/8	(		Left parenthesis
41.	2/9	)		Right parenthesis

Ordinal Position	Code	Graphic	ORD Mnemonic	Name
42.	2/10	*		Asterisk
43.	2/11	+		Plus sign
44.	2/12	,		Comma
45.	2/13	-		Minus sign
46.	2/14	.		Period
47.	2/15	/		Slant
48.	3/0	0		Digit Zero
49.	3/1	1		Digit One
50.	3/2	2		Digit Two
51.	3/3	3		Digit Three
52.	3/4	4		Digit Four
53.	3/5	5		Digit Five
54.	3/6	6		Digit Six
55.	3/7	7		Digit Seven
56.	3/8	8		Digit Eight
57.	3/9	9		Digit Nine
58.	3/10	:		Colon
59.	3/11	;		Semicolon
60.	3/12	<		Less than sign
61.	3/13	=		Equals sign
62.	3/14	>		Greater than sign
63.	3/15	?		Question mark
64.	4/0	@		Commercial at
65.	4/1	A		Capital letter A
66.	4/2	B		Capital letter B
67.	4/3	C		Capital letter C
68.	4/4	D		Capital letter D
69.	4/5	E		Capital letter E
70.	4/6	F		Capital letter F
71.	4/7	G		Capital letter G
72.	4/8	H		Capital letter H
73.	4/9	I		Capital letter I
74.	4/10	J		Capital letter J
75.	4/11	K		Capital letter K
76.	4/12	L		Capital letter L
77.	4/13	M		Capital letter M
78.	4/14	N		Capital letter N
79.	4/15	O		Capital letter O
80.	5/0	P		Capital letter P
81.	5/1	Q		Capital letter Q
82.	5/2	R		Capital letter R
83.	5/3	S		Capital letter S
84.	5/4	T		Capital letter T
85.	5/5	U		Capital letter U
86.	5/6	V		Capital letter V

Ordinal Position	Code	Graphic	ORD Mnemonic	Name
87.	5/7	W		Capital letter W
88.	5/8	X		Capital letter X
89.	5/9	Y		Capital letter Y
90.	5/10	Z		Capital letter Z
91.	5/11	[		Left bracket
92.	5/12	\		Reverse slant
93.	5/13	]		Right bracket
94.	5/14	^		Circumflex accent
95.	5/15	_	UND	Underline
96.	6/0	`	GRA	Grave accent
97.	6/1	a	LCA	Small letter a
98.	6/2	b	LCB	Small letter b
99.	6/3	c	LCC	Small letter c
100.	6/4	d	LCD	Small letter d
101.	6/5	e	LCE	Small letter e
102.	6/6	f	LCF	Small letter f
103.	6/7	g	LCG	Small letter g
104.	6/8	h	LCH	Small letter h
105.	6/9	i	LCI	Small letter i
106.	6/10	j	LCJ	Small letter j
107.	6/11	k	LCK	Small letter k
108.	6/12	l	LCL	Small letter l
109.	6/13	m	LCM	Small letter m
110.	6/14	n	LCN	Small letter n
111.	6/15	o	LCO	Small letter o
112.	7/0	p	LCP	Small letter p
113.	7/1	q	LCQ	Small letter q
114.	7/2	r	LCR	Small letter r
115.	7/3	s	LCS	Small letter s
116.	7/4	t	LCT	Small letter t
117.	7/5	u	LCU	Small letter u
118.	7/6	v	LCV	Small letter v
119.	7/7	w	LCW	Small letter w
120.	7/8	x	LCX	Small letter x
121.	7/9	y	LCY	Small letter y
122.	7/10	z	LCZ	Small letter z
123.	7/11	{	LBR	Left brace
124.	7/12	|	VLN	Vertical line
125.	7/13	}	RBR	Right brace
126.	7/14	~	TIL	Tilde
127.	7/15		DEL	Delete

Additional other-characters may occur in ordinal positions greater than 127 in the standard character set. The number of additional other-characters is implementation-defined.

Table 9. Exception Codes

This table specifies the values of the EXTYPE function corresponding to the exceptions specified in this standard. Nonfatal exceptions are designated by an exclamation-point (!). The numbers in parentheses following each exception refer to the subsections in which that exception is specified.

Overflow Errors (1000)

1001 Overflow in evaluating numeric-constant (5.1, 15.1)
1002 Overflow in evaluating numeric-expression (5.3)
1003 Overflow in evaluating numeric-supplied-function (5.4,14.7)
1004 Overflow in evaluating VAL (6.4)
1005 Overflow in evaluating numeric-array-expression (7.2)
1006 Overflow in numeric datum for (MAT) READ (10.1, 10.5)
! 1007 Overflow in numeric datum for (MAT) INPUT from terminal (10.2, 10.5)
1008 Overflow in numeric data for file input (11.4)
1009 Overflow during evaluation of DET or DOT (7.2)
1010 Too many digits declared for fixed decimal (15.1, 15.2)
1011 Overflow in fixed decimal assignment (15.1, 15.2)

1051 Overflow in evaluating string-expression (6.3)
1052 Overflow in evaluating string-array-expression (7.3)
1053 Overflow in string datum for (MAT) READ (10.1, 10.5)
! 1054 Overflow in string datum for (MAT) (LINE) INPUT (10.2, 10.5)
1105 Overflow in string datum for file input (11.4)
1106 Overflow in string assignment (6.5, 9.1, 7.3)
1201 Overflow in numeric value for process input (14.4)
1202 Overflow in numeric value from shared data or message (14.5, 14.6)
1203 Overflow in string value for process input (14.4)
1204 Overflow in string value from shared data or message (14.5, 14.6)

Underflow Errors (1500)

The following exceptions are recommended in the Remarks Sections, and are not mandated.

! 1501 Numeric constant underflow (5.1)
! 1502 Numeric expression underflow (5.3)

```
! 1503 Function value underflow (5.4)
! 1504 VAL underflow (6.4)
! 1505 Array expression underflow (7.2)
! 1506 Numeric DATA underflow (10.1)
! 1507 Numeric input underflow (10.2, 10.5)
! 1508 File numeric input underflow (11.4)
```

Subscript Errors (2000)

```
   2001 Subscript out of bounds (5.2, 6.2, 14.4, 14.5)
```

Mathematical Errors (3000)

```
   3001 Division by zero (5.3)
   3002 Negative number raised to nonintegral power (5.3)
   3003 Zero raised to negative power (5.3)
   3004 Logarithm of zero or negative number (5.4)
   3005 Square root of negative number (5.4)
   3006 Zero divisor specified for MOD or REMAINDER (5.4)
   3007 Argument of ACOS or ASIN not in range -1 ≤ x ≤ 1 (5.4)
   3008 Attempt to evaluate ANGLE(0,0) (5.4)
   3009 Attempt to invert a singular matrix, or loss of all
        significance in such attempt (7.2)
```

Uninitialized Errors (3100)

The following exceptions are recommended in the Remarks Sections, and are not mandated.

```
! 3101 Uninitialized numeric-variable (5.2)
! 3102 Uninitialized string-variable (6.2)
```

Parameter Errors (4000)

```
   4001 Argument of VAL not a numeric-constant (6.4)
   4002 Argument of CHR$ out of range (6.4)
   4003 Argument of ORD not a valid character or mnemonic (6.4)
   4004 Index of SIZE out of range (7.1)
! 4005 Index in TAB less than one (10.3)
   4006 Margin setting less than current zonewidth (10.3, 11.3)
   4007 Index of ZONEWIDTH out of range (10.3, 11.3)
   4008 Index of LBOUND out of range (7.1)
   4009 Index of UBOUND out of range (7.1)
   4010 Second argument of REPEAT$ < 0 (6.4)
```

! 4101 Set-statement CLIP neither "ON" nor "OFF" (13.1)
! 4102 Set-statement TEXT JUSTIFY illegal value (13.2)

4201 First argument of BVAL has illegal character (14.7)
4202 Value of BVAL not in precision limits (14.7)
4203 First argument of BSTR$ negative (14.7)
4204 Second argument of BVAL or BSTR$ is not 2, 8, or 16 (14.7)

4301 Parameter type or count mismatch between chain-statement and corresponding program-name-line (9.3)
4302 Mismatched dimensions between chain array parameter and corresponding formal-array (9.3)
4303 Numeric parameters passed in chain having different ARITHMETIC options (9.3)

Storage Exhausted Errors (5000)

5001 Size of redimensioned array too large (7.2, 7.3, 10.5, 11.4, 13.4)
5002 Size of transform too large for receiving array (13.5)

Matrix Errors (6000)

6001 Mismatched sizes in numeric-array-expression (7.2)
6002 Argument of DET not a square matrix (7.2)
6003 Argument of INV not a square matrix (7.2)
6004 Arguments to IDN do not specify square matrix (7.2)
6005 First index greater than second in redim, or index less than lower bound (7.2, 7.3, 10.5, 11.4)

6101 Mismatched sizes in string-array-expression (7.3)
6201 Numeric array used as transform-term not 4x4 (13.5)
6301 Mismatched sizes for array in real-time structure (14.4, 14.5, 14.6)
6401 Inconsistent dimensions for an array in an array-point-list or array-locate-object (13.3, 13.4)
6402 Size-select index out of range (13.3)

File Use Errors (7000)

7001 Channel number not in range $0 \leq c \leq$ max (11.1)
! 7002 Channel zero in OPEN, CLOSE, ERASE, or with record-setter (11.1, 11.2)
7003 Nonzero channel in OPEN already active (11.1)

7004 Inactive channel in file statement other than OPEN or ASK (11.1, 11.2, 11.3, 11.4, 11.5)
7050 Keyed file OPEN with wrong collate sequence (11.1)
7051 LENGTH not greater than zero on OPEN (11.1)
7052 Device opened as RELATIVE or KEYED file (11.1)

7100 Unrecognizable file attribute in OPEN (11.1)
71xx implementation-defined failures to provide access to file in accordance with file attribute (11.1)

7202 Record-setter with RECORD for nonrelative file (11.2)
7203 Record-setter with KEY for nonkeyed file (11.2)
7204 Record-setter SAME following DELETE, OPEN, or exception (11.2)
! 7205 Record-setter used on device without that capability (11.2)
7206 Record-setter RECORD less than one (11.2)
7207 Record-setter with exact search for null KEY value (11.2)

7301 Attempt to ERASE file not opened as OUTIN (11.1)
7302 Output not possible to INPUT file (11.3)
7303 Input not possible from OUTPUT file (11.4)
7305 Attempt to delete, rewrite, or input nonexistent record (11.4, 11.5)
7308 Attempt to write existing record (11.3)
! 7311 Attempt to erase a device without erase capability (11.1)
7312 Zonewidth or margin set for non-display file (11.3)
7313 Zonewidth or margin set for INPUT file (11.3)
7314 (MAT) WRITE to keyed file without exact search (11.3)
7315 Template used with DISPLAY or INTERNAL file (11.3, 11.4, 11.5)
7316 Accessing data in a NATIVE file without template (11.3, 11.4, 11.5)
7317 (MAT) PRINT to INTERNAL or NATIVE file (11.3)
7318 (MAT) (LINE) INPUT from INTERNAL or NATIVE file (11.4)
7320 (MAT) REWRITE or DELETE on channel zero (11.5)
7321 SKIP REST on stream file (11.4)
7322 Attempt to REWRITE or DELETE to non-OUTIN file (11.5)

7401 Attempt to trace to inactive channel (12.2)
7402 Attempt to trace to non-display-format or INPUT file (12.2)

Input-Output Errors (8000)

8001 (MAT) READ beyond end of data (10.1, 10.5)
! 8002 Too few data in input-reply (10.2, 10.5)
! 8003 Too many data in input-reply (10.2, 10.5)

8011 End-of-file encountered on input (11.4)
8012 Too few data in record (11.4)
8013 Too many data in record (11.4)

8101 Nonnumeric datum for (MAT) READ or INPUT of number from DISPLAY record (10.1, 10.5, 11.4)
! 8102 Syntactically incorrect input-reply from terminal (10.2, 10.5)
! 8103 Nonnumeric datum for (MAT) INPUT of number (10.2, 10.5)
8105 Syntactically incorrect input reply from file (11.4)
8120 Type mismatch on INTERNAL input (11.4)

8201 Invalid format-string (10.4, 10.5)
8202 No format-item in format-string for output list (10.4, 10.5)
! 8203 Format-item too short for output string (10.4)
! 8204 Exrad overflow (10.4)

8251 Syntactically incorrect template (11.3, 11.4, 11.5)
8252 Type doesn't agree with template (11.3, 11.4, 11.5)
8253 Variable-field-count doesn't start an array (11.3, 11.4, 11.5)
8254 Not enough template specifiers (11.3, 11.4, 11.5)
8255 Numeric value too large for template (11.3, 11.5)
8256 String value too large for template (11.3, 11.5)

8301 Record length exceeded on output to file (11.3, 11.5)
8302 Input from a record longer than RECSIZE (11.4)

8401 Timeout on (MAT) (LINE) INPUT (10.2, 10.5)
8402 Illegal numeric value specified for time-expression (10.2, 10.5)

The following exception is recommended in the Remarks Section, and is not mandated.

! 8106 Graphic input outside current window, device window, or device viewport (13.4)

Device Errors (9000)

9xxx implementation-defined device failures

Control Errors (10000)

10001 Index out of range, no ELSE in on-goto or on-gosub (8.2)
10002 Return without corresponding gosub or on-gosub (8.2)
10004 No case-block selected and no CASE ELSE (8.4)
10005 Attempt to chain to unavailable program (9.3)
! 10007 Break statement executed when debugging active (12.2)

Graphical Errors (11000)

11004 Attempt to change coordinates or clipping during picture-def (13.5)

! 11051 Set-statement boundaries with zero width or height (13.1)
! 11052 Viewport not in range (13.1)
! 11053 Device window not in range (13.1)
! 11054 Device viewport not in display space (13.1)
! 11056 Set-statement point style out of range (13.2)
! 11062 Set-statement line style out of range (13.2)
! 11073 Set-statement text height less than or equal to zero (13.2)
! 11085 Set-statement or array-cells color index out of range (13.2, 13.3)
! 11088 Set-statement color mix parameter not in range (13.2)

11100 Graphic-output with LINES and fewer than two points, or with AREA and fewer than three points (13.3)
11140 Graphic input device not available (13.4)
11152 Start-value in locate-statement with CHOICE or VALUE out of range (13.4)

Real-Time Errors (12000)

12001 Attempt to start activity which is not stopped (14.3)
12002 Attempt to signal event which occurred, but has not yet restarted a waiting activity (14.3)
12003 Wait-statement is executed for an event which has occurred more than once (14.3)
12004 Illegal numeric value specified for time-expression (14.3)
12005 Illegal string value specified for time-expression (14.3)
12006 Attempt to connect a connected event (14.3)
12007 Attempt to disconnent a disconnected event (14.3)

12101 Timeout of wait-statement (14.3)
12102 Timeout of process-io- or seize-statement (14.4, 14.8)
12103 Timeout of send or receive statement (14.6)

12201 Incorrect process attributes (14.2)

When an exception occurs in a program-unit and is not handled by an exception-handler in that program-unit, the exception which results at the line which invoked the program-unit shall be identified by the value 100000 plus the value specified above for the exception.

==========

Appendixes.

These Appendixes are not part of American National Standard X3.113-1987, but are included for information only.

Appendix A.
Organization of the Standard

This standard is organized into a number of sections, each of which covers a group of related features of BASIC. Each section is divided further into subsections that treat particular features of BASIC. The sub-subsections of each subsection are used as follows.

A1 Subsubsection 1. General Description

This subsection briefly describes the features of BASIC to be treated.

A2 Subsubsection 2. Syntax

The exact syntax of features of the language is described in a modified context-free grammar or Backus-Naur Form. The details of this method of syntax specification are described in 3.1.

In order to keep the syntax reasonably simple the syntax specification will allow some constructions that, strictly speaking, are not legal according to this standard; e.g., it will allow the generation of the statement

```
LET X = A(1) + A(1,2)
```

in which the array A occurs with differing numbers of subscripts. Rather than ruling such constructions out by a more complicated formal syntax, this standard instead rules them out by placing restrictions on that syntax.

The primary goal of the syntax is to define the notion of a program and its constituent parts. In addition the syntax defines several other items that are not needed for the definition of a program. These include the input-prompt, which is output to request input; the input-reply and line-input-reply, which are strings supplied in response to a request for input; and edit-commands, which may be applied to modify programs.

A3 Subsubsection 3. Examples

A short list of valid examples that can be generated by the productions in Subsection 2 is given. The numbering of the examples corresponds to the numbering of the productions, and will not be consecutive if examples are not given for all rules.

A4 Subsubsection 4. Semantics

The semantic rules in this standard assign a meaning to the constructions generated according to the syntax.

A5 Subsubsection 5. Exceptions

This subsection contains a list of those exception conditions which a standard-conforming implementation must recognize. Exception numbers (values of the EXTYPE function) are also given.

A6 Subsubsection 6. Remarks

This subsection contains remarks that point out certain features of this standard as well as remarks which make recommendations concerning the implementation of a BASIC language processor in an operating environment.

Appendix B.
Scope Rules

The scope of an entity is that part of the program where its name is recognized as referring to that object (as opposed to not being recognized at all, or recognized as referring to some other object). In general, an entity is known by only one name, and so the scope of recognition of its name is also the scope in which the object itself can be accessed. In the special case of parameter passing by reference, the same object is known by two different names, and so the object itself may be accessed outside the scope of its name.

In all cases, the indicated scope is the scope of the name of the object in question.

	Object	Scope
1.	non-parameter variable	program-unit
2.	non-parameter array	program-unit
3.	program-unit parameter	program-unit (se note)
4.	internal-proc-def parameter	internal-proc-def (see note)
5.	internal-proc-def	program-unit
6.	program-unit	program
7.	DATA	program-unit
8.	channel-number (non-zero)	program-unit
9.	channel zero	program
10.	IMAGE, TEMPLATE	program-unit
11.	GOSUB stack	smaller of program-unit, internal-proc-def, when-block, or exception-handler
12.	OPTIONs	program-unit
13.	filenames	program
14.	RND sequence	program
15.	ports (data, message, or process)	program
16.	STRUCTURE	program
17.	Graphic and PRINT set-objects	program
18.	Line-number	program-unit
19.	DEBUG and TRACE state	program-unit

NOTE: Even though the name is known only to the program-unit or or internal-proc-def, when a parameter is passed by reference, the object denoted is common to both the invoked and invoking unit.

Appendix C.
Implementation-Defined Features

A number of the features defined in this Standard have been left for definition by the implementer. However, this will not affect portability, provided that the limits recommended in the various sections are respected. The way these features are implemented shall be defined in the user or system manual of the specific implementation.

The following is a list of implementation-defined features.

Subsection 2.3

(1) Interpretation of syntactically illegal constructs
(2) Format of error messages

Subsection 2.4

(1) Format of exception messages
(2) Hardware dependent exceptions
(3) Order of exception detection in a line

Subsection 3.2

(1) Certain semantic rules for native data

Subsection 4.1

(1) Other-character
(2) Coding for the native collating sequence

Subsection 4.2

(1) End-of-line
(2) Maximum physical line length
(3) Effect of parameter list in program-name-line of program not initiated by a chain-statement
(4) Relationship of program-designator and program-name

Subsection 4.4

(1) Restrictions on identifiers for procedures compiled independently from the main program

Subsection 5.1

(1) Precision and range of numeric-constants

APPENDIX

Subsection 5.2

(1) Initial value of numeric variables

Subsection 5.3

(1) Order of evaluation of numeric-expressions

Subsection 5.4

(1) Accuracy of evaluation of numeric functions
(2) Value of MAXNUM and EPS
(3) Pseudorandom number sequence
(4) Availability of calendar and clock
(5) Time zone for DATE and TIME

Subsection 5.6

(1) Precision and range of numeric values
(2) Precision and range of floating decimal arithmetic
(3) Precision and range of native arithmetic
(4) Accuracy of evaluation of numeric expressions

Subsection 6.2

(1) Maximum length of undeclared string-variables
(2) Initial value of string-variables

Subsection 6.4

(1) Values of CHR$ for the native character set
(2) Values of ORD for the native character set
(3) Availability of calendar and clock
(4) Time-zone for DATE$ and TIME$

Subsection 6.6

(1) Collating sequence under OPTION COLLATE NATIVE
(2) Maximum length of declared string-variables without length-max

Subsection 7.1

(1) Maximum lengths of strings in string-arrays with length-max

Subsection 7.2

(1) Value of the inverse of a singular matrix

Subsection 9.1

(1) Maximum length of string parameters without length-max
(2) Value of a defined function when no value has been specified
(3) Initial values of local variables in external functions

Subsection 9.2

(1) Maximum length of string parameters without length-max
(2) Effect of redimensioning an array parameter when an element of that array is also a parameter
(3) Initial values of variables which are not formal parameters to a procedure

Subsection 9.3

(1) Interpretation of the program-designator in a chain-statement
(2) Interpretation of upper- and lower-case-letters in a program-designator
(3) Initial values of variables in a chained-to program

Subsection 10.2

(1) Input-prompt
(2) Means of requesting input in batch mode
(3) Values (minimum and maximum) and resolution of timeout-expression and time-inquiry

Subsection 10.3

(1) Effect of invoking a function which causes printing while printing
(2) Significance width for printing numeric representations
(3) Exrad width for printing numeric representations
(4) Effect of nonprinting characters on columnar position
(5) Default margin
(6) Default zonewidth
(7) Treatment of trailing space at end of print line
(8) Use of upper- or lower-case "E" in exrad

APPENDIX

Subsection 10.5

(1) Treatment of re-supply of input to a re-dimensioned array

Section 11

(1) Effect of certain combinations of file organization and type

Subsection 11.1

(1) Maximum channel number
(2) Whether a file name with different case letters (lower or upper) denotes the same file or different files
(3) Effect of attempting to open an already open file
(4) Number of channels which can be active simultaneously
(5) Attempting to open a file with attributes different from those under which it was created
(6) Attempting to reopen a file under a different ARITHMETIC option
(7) Two program-units attempting to open a file under different attributes or options
(8) Means of insuring preservation of file contents between runs
(9) Effect of certain combinations of file organization and type
(10) Length of records in INTERNAL and NATIVE files
(11) Maximum length of records when not specified or available
(12) Value of DATUM for a non-stream file
(13) Value of ask-attribute NAME for channel zero
(14) Meaning of exception codes 7101-7199
(15) Maximum length of keys for KEYED file

Subsection 11.2

(1) Method of signifying that data is not available for input on a non-file device channel

Subsection 11.3

(1) Means of indicating end-of-record
(2) Default margin and zonewidth
(3) Maximum margin and zonewidth supported
(4) Accuracy of printed numeric values produced by PRINT for DISPLAY files

Subsection 11.4

(1) Number of significant digits for value received from a numeric-field in a NATIVE file
(2) Effect of input-control-items on files and nonterminal devices
(3) Precision of numeric-constants received from DISPLAY files
(4) Precision of numeric-values that can be retrieved without loss of precision from a NATIVE file
(5) Retrieving a record from a NATIVE file having contents which are incompatible with the TEMPLATE
(6) Use of fatal or nonfatal exception procedure on illegal input-reply

Subsection 11.5

(1) Effect of data modification statements on files that are not RELATIVE or KEYED
(2) Use of SKIP in incompatible template for REWRITE

Subsection 12.1

(1) Value of EXTYPE for locally defined exceptions
(2) Format of EXTEXT$ value

Subsection 12.2

(1) Actions allowed by debugging system
(2) Form of trace reports

Subsection 13.1

(1) Manner of selecting a particular graphic display device
(2) Effect of "inverted" windows

Subsection 13.2

(1) Number of line styles available for graphics
(2) Effect of line styles other than 1, 2, 3, or 4
(3) Number of point styles available for graphics
(4) Effect of point styles other than 1, 2, 3, 4, or 5
(5) The number of color values available
(6) The color associated with each color value
(7) Effect of SET COLOR MIX on portion already drawn
(8) Default values of COLOR MIX
(9) Availability of general text orientation
(10) Orientation of text in labels

APPENDIX

Subsection 13.3

(1) Effect of plotting or filling from an array whose second dimension is greater than two
(2) Character size, style, and orientation for graphic labels
(3) Effect of partial clipping on characters

Subsection 13.4

(1) Means for notifying users of the need for graphic input
(2) Effect of attempt to supply graphic input outside current window, device window, or device viewport
(3) Effect of executing GET when current-transform is singular
(4) Default locations of indicators
(5) Effect of array-locate-object array having second subscript with more than two values

Subsection 14.1

(1) Scheduling of parallel-sections
(2) Interpretation of the urgency of parallel-sections
(3) Where execution of a parallel-section can be interrupted
(4) Initial values of variables in parallel-sections

Subsection 14.2

(1) Interpretation of the access-information for a process-object

Subsection 14.3

(1) Which of several activities waiting for an event is restarted
(2) Accuracy of timer

Subsection 14.4

(1) Exception conditions for process-objects

Subsection 14.5

(1) Getting an uninitialized data section

Subsection 14.6

(1) Which of several receive-statements receive a message
(2) Which of several send-statements send a message

Subsection 14.8

(1) Seize-item names

Subsection 15.1

(1) The maximum precision available for fixed decimal arithmetic
(2) The precision of fixed decimal expression and function evaluation
(3) The accuracy of fixed decimal expression and function evaluation
(4) Definition of "significant digits"
(5) The accessibility of an INTERNAL format file to programs having different ARITHMETIC options
(6) The accessibility of a NATIVE format file to programs having different ARITHMETIC options

Section 16

(1) The method of transition from editing to execution

Table 8

(1) The number of additional other-characters

It should be noted that implementation-defined features may cause the same program to produce different results on different implementations, for these and possibly other reasons:

(1) The logical flow of a program may be affected by the algorithm used for the pseudorandom number sequence

(2) The logical flow of a program may be affected by the value of machine infinitesimal, the value of MAXNUM, the precision for numeric values, or any combination of these three

(3) The initial value of variables may affect the logical flow of a program which contains logical errors

(4) The order of evaluation of numeric-expressions may affect the logical flow of a program

Appendix D.
Index of Syntactic Objects

This Appendix indexes all occurrences of terminal symbols and metanames in the syntax. Each reference has the form cc.s-pp, where cc.s indicates the section and subsection in which the metaname occurs and pp indicates the number of the production. (All production rules occur in subsubsection 2 in each subsection; the subsubsection number is therefore omitted from the reference.) An asterisk following a reference indicates that the metaname is defined in that production.

Example: 4.1-07 refers to section 4, subsection 4.1, subsubsection 4.1.2 (the syntax subsubsection), production rule 7.

Appendix E.
Combined List of Production Rules

```
access-information          = string-constant
access-mode                 = ACCESS (INPUT / OUTPUT / OUTIN /
                              string-expression)
actual-array                = array-name
array-assignment            = numeric-array-assignment /
                              string-array-assignment
array-cells-statement       = MAT graphic-verb CELLS comma IN
                              point-pair colon numeric-array
array-declaration           = numeric-array-declaration /
                              string-array-declaration
array-geometric-statement   = MAT graphic-verb
                              geometric-object (comma
                              size-select)? colon
                              array-point-list
array-input-statement       = MAT INPUT input-modifier-list?
                              (redim-array-list /
                              variable-length-vector) /
                              MAT INPUT channel-expression
                              input-control colon
                              (redim-array-list /
                              variable-length-vector)
array-line-input-statement  = MAT LINE INPUT
                              input-modifier-list?
                              redim-string-array-list /
                              MAT LINE INPUT channel-expression
                              input-control colon
                              redim-string-array-list
array-list                  = array-name (comma array-name)*
array-locate-object         = redim-numeric-array (comma
                              redim-numeric-array)? /
                              numeric-variable-vector comma
                              numeric-variable-vector /
                              numeric-variable-matrix
array-locate-statement      = MAT LOCATE  point-select colon
                              array-locate-object
array-name                  = numeric-array /
                              string-array
array-output-list           = array-name (comma array-name)*
array-point-list            = numeric-array (comma
                              numeric-array)?
array-print-list            = array-name (print-separator
                              array-name)* print-separator?
array-print-statement       = MAT PRINT (array-print-list /
                              (USING image colon
                              array-output-list)) /
```

```
                                MAT PRINT channel-expression
                                print-control colon
                                (array-print-list /
                                array-output-list)
array-read-statement          = MAT READ (missing-recovery colon)?
                                redim-array-list /
                                MAT READ channel-expression
                                read-control colon redim-array-list
array-rewrite-statement       = MAT REWRITE channel-expression
                                rewrite-control colon
                                array-list
array-write-statement         = MAT WRITE channel-expression
                                write-control colon array-list
ask-attribute-name            = core-attribute-name /
                                enhanced-attribute-name
ask-io-item                   = (MARGIN / ZONEWIDTH)
                                numeric-variable
ask-io-list                   = ask-io-item (comma ask-io-item)*
ask-item                      = ask-attribute-name variable
                                variable*
ask-item-list                 = ask-item (comma ask-item)*
ask-object                    = WINDOW boundary-variables /
                                VIEWPORT boundary-variables /
                                DEVICE WINDOW boundary-variables /
                                DEVICE VIEWPORT
                                boundary-variables /
                                DEVICE SIZE numeric-variable comma
                                numeric-variable comma
                                string-variable /
                                CLIP string-variable /
                                primitive-1 STYLE
                                numeric-variable /
                                primitive-2 COLOR
                                numeric-variable /
                                TEXT text-facet numeric-variable /
                                TEXT JUSTIFY string-variable comma
                                string-variable /
                                MAX primitive-1 STYLE
                                numeric-variable /
                                MAX COLOR numeric-variable /
                                COLOR MIX left-parenthesis index
                                right-parenthesis mix-list /
                                MAX device-type DEVICE
                                numeric-variable /
                                PIXEL SIZE left-parenthesis
                                point-pair right-parenthesis
                                numeric-variable comma
                                numeric-variable /
```

```
                                  PIXEL ARRAY point-location
                                  numeric-array (comma
                                  string-variable)? /
                                  PIXEL`VALUE point-location
                                  numeric-variable
ask-statement                   = ASK ask-io-list /
                                  ASK channel-setter ask-item-list /
                                  ASK ask-object status-clause?
block                           = statement-line / loop /
                                  if-block / select-block /
                                  image-line / protection-block /
                                  real-time-block
bound-argument                  = left-parenthesis actual-array
                                  (comma index)? right-parenthesis
boundaries                      = boundary comma boundary comma
                                  boundary comma boundary
boundary                        = numeric-expression
boundary-variables              = numeric-variable comma
                                  numeric-variable comma
                                  numeric-variable comma
                                  numeric-variable
bounds                          = left-parenthesis bounds-range
                                  (comma bounds-range)*
                                  right-parenthesis
bounds-range                    = signed-integer TO
                                  signed-integer / signed-integer
break-statement                 = BREAK
call-statement                  = CALL subprogram-name
                                  procedure-argument-list?
case-block                      = case-line block*
case-else-block                 = case-else-line block*
case-else-line                  = line-number CASE ELSE tail
case-item                       = constant / range
case-line                       = line-number case-statement tail
case-list                       = case-item (comma case-item)*
case-port-block                 = case-port-line block*
case-port-line                  = line-number case-port-statement
                                  tail
case-port-statement             = CASE (SEND / RECEIVE) MESSAGE
                                  message-port-name /
                                  CASE EVENT event-name
case-statement                  = CASE case-list
case-timeout-block              = case-timeout-line block*
case-timeout-line               = line-number case-timeout-statement
                                  tail
case-timeout-statement          = CASE TIMEOUT
                                  numeric-time-expression
cause-statement                 = CAUSE EXCEPTION exception-type
```

```
chain-statement             = CHAIN program-designator (WITH
                              function-arg-list)?
channel-expression          = number-sign index
channel-number              = number-sign integer
channel-setter              = channel-expression colon
character                   = quotation-mark /
                              non-quote-character
clear-statement             = CLEAR
close-statement             = CLOSE channel-expression
collate-sequence            = COLLATE (STANDARD / NATIVE /
                              string-expression)
comparison                  = numeric-expression relation
                              numeric-expression /
                              string-expression relation
                              string-expression
concatenation               = ampersand
conditional-statement       = if-statement /
                              on-gosub-statement /
                              on-goto-statement
conjunction                 = relational-term
                              (AND relational-term)*
connect-statement           = CONNECT EVENT event-list
constant                    = numeric-constant /
                              string-constant
control-transfer            = gosub-statement / goto-statement /
                              if-statement / io-recovery /
                              on-gosub-statement /
                              on-goto-statement
control-variable            = simple-numeric-variable
coordinate-pair             = numeric-expression comma
                              numeric-expression
coordinate-variables        = numeric-variable comma
                              numeric-variable
core-attribute-name         = ACCESS / DATUM / ERASABLE /
                              FILETYPE / MARGIN / NAME /
                              ORGANIZATION / POINTER / RECSIZE /
                              RECTYPE / SETTER / ZONEWIDTH
core-file-attribute         = access-mode / file-organization /
                              record-type / record-size
core-file-org-value         = SEQUENTIAL / STREAM
core-record-setter          = BEGIN / END / NEXT / SAME
core-record-type-value      = DISPLAY / INTERNAL
current-transform           = TRANSFORM
data-io-statement           = put-statement / get-statement
data-list                   = datum (comma datum)*
data-port-dec               = SHARED data-port-name
                              bounds? OF structure-name
data-port-name              = letter identifier-character*
```

```
data-statement              = DATA data-list
data-structure-dec          = STRUCTURE structure-name colon
                              repeat-count? type
                              (comma repeat-count? type)*
datum                       = constant / unquoted-string
debug-statement             = DEBUG ( ON / OFF )
declarative-statement       = data-statement /
                              declare-statement /
                              dimension-statement /
                              null-statement /
                              option-statement /
                              remark-statement /
                              template-statement
declare-statement           = DECLARE type-declaration
def-statement               = numeric-def-statement /
                              string-def-statement
def-type                    = DEF function-list
defined-function            = numeric-defined-function /
                              string-defined-function /
                              fixed-defined-function
delete-command              = DELETE segment-list
delete-control              = (comma delete-control-item)*
delete-control-item         = missing-recovery / record-setter
delete-statement            = DELETE channel-expression
                              delete-control
detached-handler            = handler-line exception-handler
                              end-handler-line
device-select               = left-parenthesis index
                              right-parenthesis
device-type                 = POINT / MULTIPOINT / CHOICE /
                              VALUE
digit                       = 0/1/2/3/4/5/6/7/8/9
digit-place                 = asterisk / number-sign /
                              percent-sign
dimension-list              = array-declaration
                              (comma array-declaration)*
dimension-statement         = DIM dimension-list
disconnect-statement        = DISCONNECT EVENT event-list
disjunction                 = conjunction (OR conjunction)*
do-body                     = block* loop-line
do-line                     = line-number do-statement tail
do-loop                     = do-line do-body
do-statement                = DO exit-condition?
double-quote                = quotation-mark quotation-mark
draw-statement              = DRAW picture-invocation (WITH
                              transform)?
e-format-item               = (i-format-item / f-format-item)
                              circumflex-accent circumflex-accent
```

```
                                circumflex-accent
                                circumflex-accent*
edit-command                  = delete-command / extract-command /
                                list-command / renumber-command
else-block                    = else-line block*
else-line                     = line-number ELSE tail
elseif-block                  = elseif-then-line block*
elseif-then-line              = line-number ELSEIF
                                relational-expression THEN tail
end-function-line             = line-number END FUNCTION tail
end-handler-line              = line-number END HANDLER
                                tail
end-if-line                   = line-number END IF tail
end-line                      = line-number end-statement tail
end-of-line                   = [implementation-defined]
end-paract-line               = line-number end-paract-statement
                                tail
end-paract-statement          = END PARACT
end-picture-line              = line-number END PICTURE tail
end-process-line              = line-number END PROCESS tail
end-seize-line                = line-number END SEIZE tail
end-select-line               = line-number END SELECT tail
end-statement                 = END
end-sub-line                  = line-number end-sub-statement tail
end-sub-statement             = END SUB
end-when-line                 = line-number END WHEN tail
enhanced-attribute-name       = RECORD / KEY / COLLATE
enhanced-file-attribute       = collate-sequence
enhanced-file-org-value       = RELATIVE / KEYED
enhanced-record-setter        = RECORD index /
                                KEY (exact-search / inexact-search)
                                string-expression
enhanced-record-type-value    = NATIVE
equality-relation             = equals-sign / not-equals
erase-statement               = ERASE REST? channel-expression
event-clause                  = EVENT event-name
event-list                    = event-name (comma event-name)*
event-name                    = letter identifier-character*
exact-search                  = equals-sign?
exception-handler             = block*
exception-type                = index
exit-condition                = (WHILE / UNTIL)
                                relational-expression
exit-do-statement             = EXIT DO
exit-for-statement            = EXIT FOR
exit-function-statement       = EXIT FUNCTION
exit-handler-statement        = EXIT HANDLER
exit-picture-statement        = EXIT PICTURE
```

```
exit-seize-statement          = EXIT SEIZE
exit-sub-statement            = EXIT SUB
expression                    = numeric-expression /
                                string-expression
expression-list               = expression (comma expression)*
exrad                         = E sign? integer
external-function-def         = external-function-line
                                unit-block* end-function-line
external-function-line        = line-number EXTERNAL FUNCTION
                                (numeric-defined-function /
                                (string-defined-function
                                length-max?))
                                function-parm-list? tail /
                                line-number EXTERNAL FUNCTION
                                fixed-defined-function
                                function-parm-list? tail
external-function-type        = EXTERNAL FUNCTION function-list
external-picture-def          = external-picture-line
                                unit-block* end-picture-line
external-picture-line         = line-number EXTERNAL
                                picture-statement tail
external-picture-type         = EXTERNAL PICTURE picture-name-list
external-sub-def              = external-sub-line unit-block*
                                end-sub-line
external-sub-line             = line-number EXTERNAL
                                sub-statement tail
external-sub-type             = EXTERNAL SUB sub-list
extract-command               = EXTRACT segment-list
f-format-item                 = period number-sign number-sign* /
                                i-format-item period number-sign*
factor                        = primary (circumflex-accent
                                primary)*
field-specifier               = numeric-specifier /
                                string-specifier
file-attribute                = core-file-attribute /
                                enhanced-file-attribute
file-attribute-list           = (comma file-attribute)*
file-name                     = string-expression
file-organization             = ORGANIZATION
                                (file-organization-value /
                                string-expression)
file-organization-value       = core-file-org-value /
                                enhanced-file-org-value
fixed-declaration             = simple-numeric-variable
                                fixed-point-type? /
                                numeric-array-declaration
                                fixed-point-type?
fixed-defined-function        = numeric-defined-function
```

```
fixed-field-count          = SKIP? (integer OF)?
fixed-formal-array         = formal-array fixed-point-type
fixed-point-size           = integer-size period? /
                             integer-size? period fraction-size
fixed-point-type           = asterisk fixed-point-size
floating-characters        = (plus-sign* / minus-sign*)
                             dollar-sign? /
                             dollar-sign* (plus-sign /
                             minus-sign)?
for-body                   = block* next-line
for-line                   = line-number for-statement tail
for-loop                   = for-line for-body
for-statement              = FOR control-variable equals-sign
                             initial-value TO limit
                             (STEP increment)?
formal-array               = array-name left-parenthesis
                             comma* right-parenthesis
format-item                = (justifier? floating-characters
                             (i-format-item / f-format-item /
                             e-format-item)) / justifier
format-string              = literal-string
                             (format-item literal-string)*
formatted-print-list       = USING image (colon output-list)?
fraction                   = period integer
fraction-size              = integer
function-arg-list          = left-parenthesis function-argument
                             (comma function-argument)*
                             right-parenthesis
function-argument          = expression / actual-array
function-def               = internal-function-def /
                             external-function-def
function-list              = defined-function
                             (comma defined-function)*
function-parameter         = simple-variable / formal-array /
                             numeric-fixed-parameter
function-parm-list         = left-parenthesis function-parameter
                             (comma function-parameter)*
                             right-parenthesis
geometric-object           = POINTS / LINES / AREA
geometric-statement        = graphic-verb geometric-object
                             colon point-list /
                             PLOT LINES / (PLOT LINES colon
                             point-list semicolon)
get-statement              = GET FROM data-port-name
                             subscript-part? TO in-structure
                             timeout-expression?
gosub-statement            = (GOSUB / GO SUB) line-number
goto-statement             = (GOTO / GO TO) line-number
```

```
graphic-input-statement      = locate-statement /
                               array-locate-statement /
                               GET point-select colon
                               coordinate-variables /
                               MAT GET point-select colon
                               array-locate-object
graphic-output-statement     = geometric-statement /
                               array-geometric-statement /
                               graphic-text-statement /
                               array-cells-statement
graphic-text-statement       = graphic-verb TEXT initial-point
                               (comma USING image colon
                               expression-list /
                               colon string-expression)
graphic-verb                 = GRAPH /
                               PLOT
handler-line                 = line-number HANDLER
                               handler-name tail
handler-name                 = routine-identifier
handler-return-statement     = RETRY / CONTINUE
i-format-item                = digit-place digit-place* (comma
                               digit-place digit-place*)*
identifier                   = numeric-identifier /
                               string-identifier /
                               routine-identifier /
                               real-time-identifier
identifier-character         = letter / digit / underline
if-block                     = if-then-line then-block
                               elseif-block* else-block?
                               end-if-line
if-clause                    = imperative-statement / line-number
if-statement                 = IF relational-expression
                               THEN if-clause (ELSE if-clause)?
if-then-line                 = line-number IF
                               relational-expression THEN tail
image                        = line-number / string-expression
image-line                   = line-number IMAGE colon
                               format-string end-of-line
imperative-statement         = array-assignment /
                               array-input-statement /
                               array-line-input-statement /
                               array-print-statement /
                               array-read-statement /
                               array-write-statement /
                               ask-statement /
                               break-statement / call-statement /
                               cause-statement / chain-statement /
                               close-statement / debug-statement /
```

```
                                erase-statement /
                                exit-do-statement /
                                exit-for-statement /
                                exit-function-statement /
                                exit-handler-statement /
                                exit-sub-statement /
                                gosub-statement / goto-statement /
                                handler-return-statement /
                                input-statement / let-statement /
                                line-input-statement /
                                numeric-function-let-statement /
                                open-statement / print-statement /
                                randomize-statement /
                                read-statement /
                                restore-statement /
                                return-statement / set-statement /
                                stop-statement /
                                string-function-let-statement /
                                trace-statement / write-statement /
                                rewrite-statement /
                                array-rewrite-statement /
                                delete-statement /
                                clear-statement /
                                graphic-output-statement /
                                graphic-input-statement  /
                                draw-statement /
                                exit-picture-statement /
                                transform-assignment
in-statement                  = IN FROM (process-port-name /
                                process-port-array subscript-part)
                                TO in-structure timeout-expression?
in-structure                  = in-structure-element
                                (comma in-structure-element)*
in-structure-element          = variable / array-name
increment                     = numeric-expression
index                         = numeric-expression
inexact-search                = greater-than-sign / not-less
initial-number                = line-number
initial-point                 = comma AT coordinate-pair
initial-value                 = numeric-expression
input-control                 = (comma input-control-item)*
input-control-item            = core-record-setter /
                                missing-recovery /
                                prompt-specifier /
                                timeout-expression / time-inquiry
input-modifier                = prompt-specifier /
                                timeout-expression /
                                time-inquiry
```

```
input-modifier-list          = input-modifier
                               (comma input-modifier)* colon
input-prompt                 = [implementation-defined]
input-reply                  = data-list comma? end-of-line
input-statement              = INPUT input-modifier-list?
                               variable-list / INPUT
                               channel-expression input-control
                               colon variable-list
                               (comma SKIP REST)?
integer                      = digit digit*
integer-size                 = integer
internal-def-line            = line-number def-statement tail
internal-function-def        = internal-def-line /
                               internal-function-line
                               block* end-function-line
internal-function-line       = line-number FUNCTION
                               (numeric-defined-function /
                               (string-defined-function
                               length-max?))
                               function-parm-list? tail /
                               line-number FUNCTION
                               fixed-defined-function
                               function-parm-list? tail
internal-function-type       = FUNCTION function-list
internal-picture-def         = internal-picture-line
                               block* end-picture-line
internal-picture-line        = line-number picture-statement tail
internal-picture-type        = PICTURE picture-name-list
internal-proc-def            = internal-function-def /
                               internal-sub-def /
                               detached-handler /
                               internal-picture-def
internal-sub-def             = internal-sub-line block*
                               end-sub-line
internal-sub-line            = line-number sub-statement tail
internal-sub-type            = SUB sub-list
io-qualifier                 = INPUT / OUTPUT / OUTIN
io-recovery                  = missing-recovery /
                               not-missing-recovery
io-recovery-action           = exit-do-statement /
                               exit-for-statement / line-number
justifier                    = greater-than-sign / less-than-sign
length-max                   = asterisk integer
let-statement                = numeric-let-statement /
                               string-let-statement
letter                       = upper-case-letter /
                               lower-case-letter
limit                        = numeric-expression
```

```
line                        = case-line / case-else-line /
                              do-line / else-line /
                              elseif-then-line /
                              end-function-line /
                              end-handler-line /
                              end-if-line / end-line /
                              end-select-line / end-sub-line /
                              end-when-line /
                              external-function-line /
                              external-sub-line /
                              for-line / handler-line /
                              internal-def-line /
                              internal-function-line /
                              internal-sub-line /
                              if-then-line / image-line /
                              loop-line / next-line /
                              program-name-line / remark-line /
                              select-line / statement-line /
                              use-line /
                              when-line / when-use-name-line /
                              end-picture-line /
                              external-picture-line /
                              internal-picture-line /
                              paract-line / end-paract-line /
                              rt-declare-line /
                              process-declare-line /
                              process-element-line /
                              end-process-line /
                              seize-line / end-seize-line
line-continuation           = ampersand space* tail ampersand
line-input-reply            = character* end-of-line
line-input-statement        = LINE INPUT input-modifier-list?
                              string-variable-list /
                              LINE INPUT channel-expression
                              input-control colon
                              string-variable-list
line-number                 = digit digit*
list-command                = LIST segment-list?
literal-item                = letter / digit /
                              apostrophe / colon / equals-sign /
                              exclamation-point /
                              left-parenthesis / question-mark /
                              right-parenthesis / semicolon /
                              slant / space / underline
literal-string              = literal-item*
locate-statement            = LOCATE (point-select colon
                              coordinate-variables /
                              value-select colon
```

```
                                numeric-variable)
loop                          = do-loop / for-loop
loop-line                     = line-number loop-statement tail
loop-statement                = LOOP exit-condition?
lower-case-letter             = a/b/c/d/e/f/g/h/i/j/k/l/m/
                                n/o/p/q/r/s/t/u/v/w/x/y/z
main-program                  = unit-block* end-line
maxsize-argument              = left-parenthesis actual-array
                                right-parenthesis
message-io-statement          = send-statement / receive-statement
message-port-dec              = MESSAGE message-port-name
                                OF structure-name
message-port-name             = letter identifier-character*
missing-recovery              = IF MISSING THEN io-recovery-action
mix-list                      = numeric-variable comma
                                numeric-variable comma
                                numeric-variable
multiplier                    = asterisk / slant
next-line                     = line-number next-statement tail
next-statement                = NEXT control-variable
non-quote-character           = ampersand / apostrophe /
                                asterisk / circumflex-accent /
                                colon / comma / dollar-sign /
                                equals-sign /
                                exclamation-point /
                                greater-than-sign /
                                left-parenthesis /
                                less-than-sign / number-sign /
                                percent-sign / question-mark /
                                right-parenthesis / semicolon /
                                slant / underline /
                                unquoted-string-character
not-equals                    = less-than-sign greater-than-sign /
                                greater-than-sign less-than-sign
not-greater                   = less-than-sign equals-sign /
                                equals-sign less-than-sign
not-less                      = greater-than-sign equals-sign /
                                equals-sign greater-than-sign
not-missing-recovery          = IF THERE THEN io-recovery-action
null-statement                = tail-comment
numeric-array                 = numeric-identifier
numeric-array-assignment      = MAT numeric-array equals-sign
                                numeric-array-expression
numeric-array-declaration     = numeric-array bounds
numeric-array-element         = numeric-array subscript-part
numeric-array-expression      = (numeric-array
                                numeric-array-operator)?
                                numeric-array /
```

```
                                scalar-multiplier
                                numeric-array /
                                numeric-array-value /
                                numeric-array-function-ref
numeric-array-function-ref    = (TRN / INV) left-parenthesis
                                numeric-array
                                right-parenthesis
numeric-array-operator        = sign / asterisk
numeric-array-value           = scalar-multiplier?
                                (CON / IDN / ZER) redim? /
                                TRANSFORM
numeric-constant              = sign? numeric-rep
numeric-declaration           = simple-numeric-variable /
                                numeric-array-declaration
numeric-def-statement         = DEF numeric-defined-function
                                function-parm-list? equals-sign
                                numeric-expression /
                                DEF fixed-defined-function
                                function-parm-list? equals-sign
                                numeric-expression
numeric-defined-function      = numeric-identifier
numeric-expression            = sign? term (sign term)*
numeric-field-size            = fixed-point-size / E
numeric-fixed-parameter       = simple-numeric-variable
                                fixed-point-type /
                                fixed-formal-array
numeric-function              = numeric-defined-function /
                                numeric-supplied-function
numeric-function-let-statement
                              = LET numeric-defined-function
                                equals-sign numeric-expression
numeric-function-ref          = numeric-function
                                function-arg-list? /
                                MAXLEN left-parenthesis
                                (simple-string-variable /
                                string-array) right-parenthesis /
                                MAXSIZE maxsize-argument /
                                SIZE bound-argument /
                                LBOUND bound-argument /
                                UBOUND bound-argument /
                                DET (left-parenthesis
                                numeric-array
                                right-parenthesis) /
                                DOT left-parenthesis
                                numeric-array comma
                                numeric-array
                                right-parenthesis
numeric-identifier            = letter identifier-character*
```

```
numeric-let-statement          = LET numeric-variable-list
                                 equals-sign numeric-expression
numeric-rep                    = significand exrad?
numeric-specifier              = NUMERIC asterisk numeric-field-size
numeric-supplied-function      = ABS / ACOS / ANGLE / ASIN /
                                 ATN / CEIL / COS / COSH / COT /
                                 CSC / DATE / DEG / EPS / EXP /
                                 FP / MAXNUM / INT / IP / LOG /
                                 LOG10 / LOG2 / MAX / MIN / MOD /
                                 PI / RAD / REMAINDER / RND /
                                 ROUND / SEC / SGN / SIN / SINH /
                                 SQR / TAN / TANH / TIME /
                                 TRUNCATE / LEN / ORD / POS /
                                 VAL / EXLINE / EXTYPE / BVAL
numeric-time-expression        = numeric-expression
numeric-type                   = NUMERIC numeric-declaration
                                 (comma numeric-declaration)* /
                                 NUMERIC fixed-point-type?
                                 fixed-declaration
                                 (comma fixed-declaration)*
numeric-variable               = simple-numeric-variable /
                                 numeric-array-element
numeric-variable-list          = numeric-variable
                                 (comma numeric-variable)*
numeric-variable-matrix        = numeric-array left-parenthesis
                                 question-mark comma
                                 right-parenthesis
numeric-variable-vector        = numeric-array left-parenthesis
                                 question-mark right-parenthesis
on-gosub-statement             = ON index (GOSUB / GO SUB)
                                 line-number (comma line-number)*
                                 (ELSE imperative-statement)?
on-goto-statement              = ON index (GOTO / GO TO)
                                 line-number (comma line-number)*
                                 (ELSE imperative-statement)?
open-statement                 = OPEN channel-setter NAME file-name
                                 file-attribute-list
option                         = ARITHMETIC (DECIMAL / NATIVE) /
                                 ANGLE (DEGREES / RADIANS) /
                                 COLLATE (NATIVE / STANDARD) /
                                 BASE ( 0 / 1 ) /
                                 ARITHMETIC FIXED fixed-point-type
option-list                    = option (comma option)*
option-statement               = OPTION option-list
other-character                = [implementation-defined]
out-statement                  = OUT TO (process-port-name /
                                 process-port-array subscript-part)
                                 FROM out-structure
```

```
                                  timeout-expression?
out-structure                   = out-structure-element
                                  (comma out-structure-element)*
out-structure-element           = expression / array-name
output-list                     = expression (comma expression)*
                                  semicolon?
paract-line                     = line-number paract-statement tail
paract-statement                = PARACT routine-identifier
                                  (URGENCY urgency)?
parallel-section                = remark-line* paract-line
                                  block* end-paract-line
parstop-statement               = PARSTOP
picture-def                     = internal-picture-def /
                                  external-picture-def
picture-invocation              = picture-name
                                  procedure-argument-list?
picture-name                    = routine-identifier
picture-name-list               = picture-name (comma picture-name)*
picture-statement               = PICTURE picture-name
                                  procedure-parm-list?
plain-string-character          = digit / letter / period /
                                  plus-sign / minus-sign
point-list                      = coordinate-pair (semicolon
                                  coordinate-pair)*
point-location                  = left-parenthesis coordinate-pair
                                  right-parenthesis
point-pair                      = coordinate-pair semicolon
                                  coordinate-pair
point-select                    = POINT device-select? initial-point?
pointer-control                 = POINTER core-record-setter /
                                  enhanced-record-setter
pointer-items                   = (pointer-control / io-recovery /
                                  pointer-control comma io-recovery)
primary                         = numeric-rep / numeric-variable /
                                  numeric-function-ref /
                                  left-parenthesis numeric-expression
                                  right-parenthesis
primitive-1                     = POINT / LINE
primitive-2                     = primitive-1 / TEXT / AREA
print-control                   = (comma print-control-item)*
print-control-item              = core-record-setter /
                                  not-missing-recovery / USING image
print-item                      = expression / tab-call
print-list                      = (print-item? print-separator)*
                                  print-item?
print-separator                 = comma / semicolon
print-statement                 = PRINT print-list /
                                  PRINT formatted-print-list /
```

```
                                  PRINT channel-expression
                                  print-control (colon
                                  (print-list / output-list))?
procedure                       = external-function-def /
                                  external-sub-def /
                                  external-picture-def
procedure-argument              = expression / actual-array /
                                  channel-expression
procedure-argument-list         = left-parenthesis
                                  procedure-argument
                                  (comma procedure-argument)*
                                  right-parenthesis
procedure-parameter             = simple-variable / formal-array /
                                  channel-number /
                                  numeric-fixed-parameter
procedure-parm-list             = left-parenthesis
                                  procedure-parameter
                                  (comma procedure-parameter)*
                                  right-parenthesis
procedure-part                  = remark-line* procedure
process-array-dec               = process-declare-line
                                  process-element-line*
                                  end-process-line
process-clause                  = io-qualifier process-port-name
                                  OF structure-name
process-declare-line            = line-number
                                  process-declare-statement tail
process-declare-statement       = DECLARE PROCESS io-qualifier
                                  process-port-array bounds
                                  OF structure-name
process-element-line            = line-number
                                  process-element-statement tail
process-element-statement       = process-port-array
                                  left-parenthesis signed-integer
                                  (comma signed-integer)?
                                  right-parenthesis colon
                                  access-information
process-io-statement            = in-statement / out-statement
process-port-array              = letter identifier-character*
process-port-dec                = PROCESS
                                  (process-clause / event-clause)
                                  access-information?
process-port-name               = letter identifier-character*
program                         = program-name-line?  main-program
                                  procedure-part* /
                                  program-name-line?
                                  real-time-program
program-designator              = string-expression
```

```
program-line                = line-number
                              (character / line-continuation)*
                              end-of-line
program-name                = routine-identifier
program-name-line           = line-number PROGRAM program-name
                              function-parm-list?  tail
program-unit                = main-program / procedure /
                              parallel-section
prompt-specifier            = PROMPT string-expression
protection-block            = when-use-block /
                              when-use-name-block
put-statement               = PUT TO data-port-name
                              subscript-part?
                              FROM out-structure
                              timeout-expression?
quoted-string               = quotation-mark
                              quoted-string-character*
                              quotation-mark
quoted-string-character     = double-quote /
                              non-quote-character
randomize-statement         = RANDOMIZE
range                       = (constant TO / IS relation)
                              constant
range-select                = comma RANGE numeric-expression TO
                              numeric-expression
read-control                = (comma read-control-item)*
read-control-item           = record-setter / missing-recovery /
                              template-identifier
read-statement              = READ (missing-recovery colon)?
                              variable-list / READ
                              channel-expression read-control
                              colon variable-list
                              (comma SKIP REST)?
real-time-block             = select-port-block / seize-block
real-time-declarations      = (remark-line / rt-declare-line /
                              process-array-dec)*
real-time-identifier        = structure-name / event-name /
                              process-port-name /
                              process-port-array /
                              data-port-name / message-port-name
real-time-program           = real-time-declarations
                              parallel-section parallel-section*
                              procedure-part*
real-time-statement         = parstop-statement /
                              scheduling-statement /
                              process-io-statement /
                              data-io-statement /
                              message-io-statement /
```

```
                                exit-seize-statement
receive-statement             = RECEIVE FROM message-port-name
                                TO in-structure
                                timeout-expression?
record-setter                 = core-record-setter /
                                enhanced-record-setter
record-size                   = RECSIZE (VARIABLE /
                                string-expression) (LENGTH index)?
record-type                   = RECTYPE (record-type-value /
                                string-expression)
record-type-value             = core-record-type-value /
                                enhanced-record-type-value
redim                         = left-parenthesis redim-bounds
                                (comma redim-bounds)*
                                right-parenthesis
redim-array                   = array-name redim?
redim-array-list              = redim-array
                                (comma redim-array)*
redim-bounds                  = (index TO)? index
redim-numeric-array           = numeric-array redim?
redim-string-array            = string-array redim?
redim-string-array-list       = redim-string-array
                                (comma redim-string-array)*
relation                      = equality-relation /
                                greater-than-sign /
                                less-than-sign /
                                not-greater / not-less
relational-expression         = disjunction
relational-primary            = comparison / left-parenthesis
                                relational-expression
                                right-parenthesis
relational-term               = NOT? relational-primary
remark-line                   = line-number (null-statement /
                                remark-statement) end-of-line
remark-statement              = REM remark-string
remark-string                 = character*
renumber-command              = RENUMBER segment-specifier?
                                renumber-parameters?
renumber-parameters           = AT initial-number
                                (STEP step-size)? / STEP step-size
                                (AT initial-number)?
repeat-count                  = integer OF
restore-statement             = RESTORE line-number?
return-statement              = RETURN
rewrite-control               = (comma rewrite-control-item)*
rewrite-control-item          = missing-recovery / record-setter /
                                template-identifier
rewrite-statement             = REWRITE channel-expression
```

```
                                rewrite-control colon
                                expression-list
rgb-list                    = numeric-expression comma
                                numeric-expression comma
                                numeric-expression
routine-identifier          = letter identifier-character*
rt-declare-line             = line-number rt-declare-statement
                                tail
rt-declare-statement        = DECLARE (data-structure-dec /
                                process-port-dec /
                                data-port-dec / message-port-dec)
scalar-multiplier           = primary asterisk
scheduling-statement        = start-statement / wait-statement /
                                signal-statement /
                                connect-statement /
                                disconnect-statement
segment-item                = line-number / FIRST / LAST
segment-list                = segment-specifier
                                (comma segment-specifier)*
segment-specifier           = segment-item (TO segment-item)?
seize-block                 = seize-line block* end-seize-line
seize-item                  = SHARED data-port-name /
                                [implementation-defined]
seize-line                  = line-number seize-statement tail
seize-list                  = seize-item (comma seize-item)*
seize-statement             = SEIZE seize-list
                                timeout-expression?
select-block                = select-line remark-line* case-block
                                case-block* case-else-block?
                                end-select-line
select-line                 = line-number select-statement tail
select-port-block           = select-port-line remark-line*
                                case-port-block case-port-block*
                                case-timeout-block? end-select-line
select-port-line            = line-number select-port-statement
                                tail
select-port-statement       = SELECT ON PORT
select-statement            = SELECT CASE expression
send-statement              = SEND TO message-port-name
                                FROM out-structure
                                timeout-expression?
set-object                  = (MARGIN / ZONEWIDTH) index /
                                channel-setter pointer-items /
                                channel-setter
                                (MARGIN / ZONEWIDTH) index /
                                WINDOW boundaries /
                                VIEWPORT boundaries /
                                DEVICE WINDOW boundaries /
```

```
                                  DEVICE VIEWPORT boundaries /
                                  CLIP string-expression /
                                  primitive-1 STYLE index /
                                  primitive-2 COLOR index /
                                  TEXT text-facet
                                  numeric-expression /
                                  TEXT JUSTIFY string-expression
                                  comma string-expression /
                                  COLOR MIX left-parenthesis index
                                  right-parenthesis rgb-list
set-statement                   = SET set-object
sign                            = plus-sign / minus-sign
signal-statement                = SIGNAL event-name
signed-integer                  = sign? integer
significand                     = integer period? / integer? fraction
simple-numeric-variable         = numeric-identifier
simple-string-declaration       = simple-string-variable length-max?
simple-string-variable          = string-identifier
simple-variable                 = simple-numeric-variable /
                                  simple-string-variable
size-select                     = LIMIT index
start-statement                 = START routine-identifier
start-value                     = comma AT numeric-expression
statement                       = declarative-statement /
                                  imperative-statement /
                                  conditional-statement /
                                  real-time-statement
statement-line                  = line-number statement tail
status-clause                   = STATUS numeric-variable
step-size                       = integer
stop-statement                  = STOP
string-array                    = string-identifier
string-array-assignment         = MAT string-array
                                  substring-qualifier?
                                  equals-sign
                                  string-array-expression
string-array-declaration        = string-array bounds
string-array-element            = string-array subscript-part
string-array-expression         = string-array-primary
                                  (concatenation
                                  string-array-primary)? /
                                  string-primary
                                  concatenation
                                  string-array-primary /
                                  string-array-primary
                                  concatenation
                                  string-primary /
                                  string-array-value
```

```
string-array-primary          = string-array
                                substring-qualifier?
string-array-value            = (string-primary
                                concatenation)?
string-constant               = quoted-string
string-declaration            = simple-string-declaration /
                                string-array-declaration
                                length-max?
string-def-statement          = DEF string-defined-function
                                length-max? function-parm-list?
                                equals-sign string-expression
string-defined-function       = string-identifier
string-expression             = string-primary
                                (concatenation string-primary)*
string-field-size             = integer
string-function               = string-defined-function /
                                string-supplied-function
string-function-let-statement= LET string-defined-function
                                equals-sign string-expression
string-function-ref           = string-function function-arg-list?
string-identifier             = letter identifier-character*
                                dollar-sign
string-let-statement          = LET string-variable-list
                                equals-sign string-expression
string-primary                = string-constant /
                                string-variable /
                                string-function-ref /
                                left-parenthesis string-expression
                                right-parenthesis
string-specifier              = STRING asterisk string-field-size
string-supplied-function      = (CHR / DATE / LCASE / LTRIM /
                                REPEAT / RTRIM / STR / TIME /
                                UCASE / USING) dollar-sign /
                                EXTEXT dollar-sign /
                                BSTR dollar-sign
string-time-expression        = string-expression
string-type                   = STRING length-max?
                                string-declaration
                                (comma string-declaration)*
string-variable               = (simple-string-variable /
                                string-array-element)
                                substring-qualifier?
string-variable-list          = string-variable
                                (comma string-variable)*
structure-name                = letter identifier-character*
sub-list                      = subprogram-name (comma
                                subprogram-name)*
sub-statement                 = SUB subprogram-name
```

```
                                  procedure-parm-list?
subprogram-def                  = internal-sub-def /
                                  external-sub-def
subprogram-name                 = routine-identifier
subscript                       = index
subscript-part                  = left-parenthesis subscript
                                  (comma subscript)*
                                  right-parenthesis
substring-qualifier             = left-parenthesis index colon
                                  index right-parenthesis
tab-call                        = TAB left-parenthesis index
                                  right-parenthesis
tail                            = tail-comment? end-of-line
tail-comment                    = exclamation-point remark-string
template-element                = fixed-field-count
                                  (field-specifier /
                                  left-parenthesis
                                  template-element-list
                                  right-parenthesis) /
                                  variable-field-count
                                  field-specifier
template-element-list           = template-element
                                  (comma template-element)*
template-identifier             = WITH (line-number /
                                  string-expression)
template-statement              = TEMPLATE colon
                                  template-element-list
term                            = factor (multiplier factor)*
text-facet                      = HEIGHT / ANGLE
then-block                      = block*
time-expression                 = numeric-time-expression /
                                  string-time-expression
time-inquiry                    = ELAPSED numeric-variable
timeout-expression              = TIMEOUT numeric-time-expression
trace-statement                 = TRACE ON (TO channel-expression)? /
                                  TRACE OFF
transform                       = transform-term (asterisk
                                  transform-term)*
transform-assignment            = MAT numeric-array equals-sign
                                  transform
transform-function              = ROTATE / SHEAR / SHIFT / SCALE
transform-term                  = transform-function
                                  function-arg-list /
                                  numeric-array / current-transform
type                            = (NUMERIC fixed-point-type? /
                                  STRING) bounds?
type-declaration                = numeric-type / string-type /
                                  def-type /
```

```
                                internal-function-type /
                                external-function-type /
                                internal-sub-type /
                                external-sub-type /
                                internal-picture-type /
                                external-picture-type
unit-block                    = internal-proc-def / block
unquoted-string               = plain-string-character /
                                plain-string-character
                                unquoted-string-character*
                                plain-string-character
unquoted-string-character     = space / plain-string-character
unsorted-program              = program-line*
upper-case-letter             = A/B/C/D/E/F/G/H/I/J/K/L/M/
                                N/O/P/Q/R/S/T/U/V/W/X/Y/Z
urgency                       = integer
use-line                      = line-number USE tail
value-select                  = CHOICE device-select?
                                start-value? /
                                VALUE device-select? range-select?
                                start-value?
variable                      = numeric-variable /
                                string-variable
variable-field-count          = question-mark OF
variable-length-vector        = array-name left-parenthesis
                                question-mark right-parenthesis
variable-list                 = variable (comma variable)*
wait-event                    = EVENT event-name
                                timeout-expression?
wait-interval                 = DELAY numeric-time-expression
wait-statement                = WAIT (wait-time / wait-interval /
                                wait-event )
wait-time                     = TIME time-expression
when-block                    = block*
when-line                     = line-number WHEN EXCEPTION IN tail
when-use-block                = when-line when-block
                                use-line exception-handler
                                end-when-line
when-use-name-block           = when-use-name-line
                                when-block end-when-line
when-use-name-line            = line-number WHEN EXCEPTION USE
                                handler-name tail
write-control                 = (comma write-control-item)*
write-control-item            = record-setter /
                                not-missing-recovery /
                                template-identifier
write-statement               = WRITE channel-expression
                                write-control colon expression-list
```

Appendix F.
Binding of GKS Level 0b to BASIC

F1. Introduction. This Appendix provides the syntax description of the language binding the Graphical Kernel System (GKS) level 0b to Basic. The semantics are described in ANSI X3.124-1985 or ISO 7942-1985 Graphics Kernel System. Parts of the syntax (and semantics) are also described in Section 13 of this standard; such places are noted following the GKS function. The matching of the parameters to those in ANSI X3.124-1985 or ISO 7942-1985 is in an obvious manner. This Appendix provides one possible method for those requiring more graphics facilities than those provided by Section 13. This Appendix should be read in conjunction with ANSI X3.124-1985 or ISO 7942-1985.

F2. GKS Control Functions

OPEN GKS

```
   gks-statement     > GRAPHICS START
```

CLOSE GKS

```
   gks-statement     > GRAPHICS STOP
```

OPEN WORKSTATION

```
   gks-statement     > GRAPHICS OPEN wkstn NAME file-name comma
                       wkstn-type-set
   wkstn             = wkstn-id colon
   wkstn-id          = number-sign index
   wkstn-type-set    = TYPE index
```

CLOSE WORKSTATION

```
   gks-statement     > GRAPHICS CLOSE wkstn-id
```

ACTIVATE WORKSTATION

```
   gks-statement     > GRAPHICS ACTIVATE wkstn-id
```

DEACTIVATE WORKSTATION

```
   gks-statement     > GRAPHICS DEACTIVATE wkstn-id
```

CLEAR WORKSTATION (See Section 13)

```
gks-statement    > CLEAR (wkstn string-expression)?
```

After the letters of string-expression have been converted to upper case it must have either the value "CONDITIONALLY" or "ALWAYS". If the optional parameters are not present, #0 and "ALWAYS" are used.

EMERGENCY CLOSE GKS

```
gks-statement    > GRAPHICS ABORT
```

UPDATE WORKSTATION

```
gks-statement    > GRAPHICS UPDATE wkstn? string-expression
```

After the letters of string-expression have been converted to upper case it must have either the value "PERFORM" or "SUPPRESS".

ESCAPE

```
gks-statement    > GRAPHICS ESCAPE expression-list
```

F3. Output Functions

POLYLINE (See Section 13)

```
gks-statement    > GRAPH LINES colon point-list /
                   MAT GRAPH LINES (comma size-select)?
                   colon array-point-list
```

POLYMARKER (See Section 13)

```
gks-statement    > GRAPH POINTS colon point-list /
                   MAT GRAPH POINTS (comma size-select)?
                   colon array-point-list
```

TEXT (See Section 13)

```
gks-statement    > GRAPH TEXT initial-point (comma USING image
                   colon expression-list / colon
                   string-expression)
```

FILL AREA (See Section 13)

```
   gks-statement     > GRAPH AREA colon point-list / MAT GRAPH AREA
                       (comma size-select)? colon array-point-list
```

CELL ARRAY (See Section 13)

```
   gks-statement     > MAT GRAPH CELLS comma IN point-pair colon
                       numeric-array
```

GENERALIZED DRAWING PRIMITIVE

```
   gks-statement     > GRAPH GDP index colon point-list FROM
                       data-record / MAT GRAPH GDP index
                       (comma size-select)? colon array-point-list
                       FROM data-record
   data-record       = numeric-array comma string-array
```

Index is the GDP identifier. Data-record is the GDP record.

F4. Output Attributes

SET POLYLINE INDEX

```
   gks-statement     > SET LINE INDEX index
```

SET LINETYPE (See Section 13)

```
   gks-statement     > SET LINE STYLE index
```

SET LINE WIDTH SCALE FACTOR

```
   gks-statement     > SET LINE SIZE numeric-expression
```

SET POLYLINE COLOUR INDEX (See Section 13)

```
   gks-statement     > SET LINE COLOR index
```

SET POLYMARKER INDEX

```
   gks-statement     > SET POINT INDEX index
```

SET MARKER TYPE (See Section 13)

```
   gks-statement     > SET POINT STYLE index
```

SET MARKER SIZE SCALE FACTOR

gks-statement > SET POINT SIZE numeric-expression

SET POLYMARKER COLOUR INDEX (See Section 13)

gks-statement > SET POINT COLOR index

SET TEXT INDEX

gks-statement > SET TEXT INDEX index

SET TEXT FONT AND PRECISION

gks-statement > SET FONT index WITH string-expression

Numeric-expression is the font number. After the letters of string-expression have been converted to upper case it must have one of the values "STRING", "CHAR", or "STROKE".

SET TEXT COLOUR INDEX (See Section 13)

gks-statement > SET TEXT COLOR index

SET CHARACTER EXPANSION FACTOR

gks-statement > SET TEXT EXPAND numeric-expression

SET CHARACTER SPACING

gks-statement > SET TEXT SPACE numeric-expression

SET CHARACTER HEIGHT (See Section 13)

gks-statement > SET TEXT HEIGHT numeric-expression

SET CHARACTER UP VECTOR (See Section 13)

gks-statement > SET TEXT ANGLE numeric-expression

SET TEXT PATH

gks-statement > SET TEXT PATH string-expression

String-expression must be one of "RIGHT", "LEFT", "UP", or "DOWN" after the letters of string-expression have been converted to upper case.

SET TEXT ALIGNMENT (See Section 13)

```
gks-statement    > SET TEXT JUSTIFY string-expression comma
                   string expression
```

SET FILL AREA INDEX

```
gks-statement    > SET AREA INDEX index
```

SET FILL AREA INTERIOR STYLE

```
gks-statement    > SET AREA STYLE string-expression
```

String-expression must be one of "HOLLOW", "SOLID", "PATTERN", or "HATCH" after its letters have been converted to upper case.

SET FILL AREA STYLE INDEX

```
gks-statement    > SET AREA STYLE INDEX index
```

SET FILL AREA COLOUR INDEX (See Section 13)

```
gks-statement    > SET AREA COLOR index
```

SET PATTERN SIZE

```
gks-statement    > SET PATTERN SIZE numeric-expression comma
                   numeric expression
```

SET PATTERN REFERENCE POINT

```
gks-statement    > SET PATTERN REF coordinate-pair
```

SET ASPECT SOURCE FLAGS

```
gks-statement    > SET FLAGS flag-type string-expression
flag-type        = LINE STYLE / LINE SIZE / LINE COLOR /
                   POINT STYLE / POINT SIZE / POINT COLOR /
                   TEXT FONT / TEXT EXPAND / TEXT SPACE /
                   TEXT COLOR / AREA STYLE /
                   AREA STYLE INDEX / AREA COLOR
```

String-expression must be one of "BUNDLED" or "INDIVIDUAL" when the letters of string-expression have been converted to upper case.

SET COLOUR REPRESENTATION (See Section 13)

```
gks-statement    > SET COLOR MIX wkstn? table-index rgb-list
table-index      = left-parenthesis index right-parenthesis
```

F5. Transformation Functions

SET WINDOW (See Section 13)

```
gks-statement    > SET WINDOW tran-number? boundaries
tran-number      = TRANSFORMATION index
```

If tran-number is not specified, 1 is assumed.

SET VIEWPORT (See Section 13)

```
gks-statement    > SET VIEWPORT tran-number? boundaries
```

If tran-number is not specified, 1 is assumed.

SELECT NORMALISATION TRANSFORMATION

```
gks-statement    > SET tran-number
```

SET CLIPPING INDICATOR (See Section 13)

```
gks-statement    > SET CLIP string-expression
```

SET WORKSTATION WINDOW (See Section 13)

```
gks-statement    > SET DEVICE WINDOW wkstn? boundaries
```

SET WORKSTATION VIEWPORT (See Section 13)

```
gks-statement    > SET DEVICE VIEWPORT wkstn? boundaries
```

F6. Metafile Functions

WRITE ITEM TO GKSM

```
gks-statement    > GRAPHICS WRITE wkstn-id TYPE index colon
                   expression-list
```

GET ITEM TYPE FROM GKSM

```
gks-statement     > GRAPHICS READ TYPE wkstn numeric-variable
```

Numeric-variable is the item type.

READ ITEM FROM GKSM

```
gks-statement     > GRAPHICS READ ITEM wkstn string-array
```

INTERPRET ITEM

```
gks-statement     > GRAPHICS DO ITEM string-array-expression
```

F7. Enquiry Functions

The error indicator (of GKS) is returned by a status-clause on an ask-statement.

INQUIRE CLIPPING INDICATOR (See Section 13)

```
gks-statement     > ASK CLIP string-variable
```

INQUIRE COLOUR FACILITIES (See Section 13)

```
gks-statement      > ASK MAX COLOR wkstn-type-select?
                     numeric-variable (comma string-variable
                     comma numeric-variable)?
wkstn-type-select = left-parenthesis wkstn-type
                     right-parenthesis
wkstn-type        = index
```

String-variable is set to either "MONOCHROME" or "COLOR".

INQUIRE COLOUR REPRESENTATION (See Section 13)

```
gks-statement     > ASK COLOR MIX wkstn? table-index mix-list
```

INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES (See Section 13)

```
gks-statement     > ASK attribute-name1 numeric-variable /
                    ASK AREA STYLE string-variable /
                    ASK TEXT FONT numeric-variable comma
                    string-variable
attribute-name1   = LINE STYLE / LINE SIZE / LINE COLOR/
                    POINT STYLE / POINT SIZE / POINT COLOR /
                    TEXT EXPAND / TEXT SPACE / TEXT COLOR /
                    AREA STYLE INDEX / AREA COLOR
```

```
INQUIRE CURRENT NORMALISATION TRANSFORMATION NUMBER

   gks-statement     > ASK TRANSFORMATION numeric-variable

INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES (See Section 13)

   gks-statement     > ASK attribute-name2 numeric-variable /
                       ASK TEXT JUSTIFY string-variable
                       comma string-variable /
                       ASK TEXT PATH string-variable /
                       ASK attribute-name3 numeric-variable
                       comma numeric-variable
   attribute-name2   = LINE INDEX / POINT INDEX / TEXT INDEX /
                       TEXT HEIGHT / TEXT ANGLE / AREA INDEX
   attribute-name3   = PATTERN SIZE / PATTERN REF

INQUIRE FILL AREA FACILITIES

   gks-statement     > ASK AREA TYPES wkstn-type-select?
                       numeric-variable comma
                       string-variable-vector comma
                       numeric-variable comma
                       numeric-variable-vector comma
                       numeric-variable

INQUIRE GENERALIZED DRAWING PRIMITIVE

   gks-statement     > ASK GDP left-parenthesis wkstn-type comma
                       index right-parenthesis string-array

INQUIRE LEVEL OF GKS

   gks-statement     > ASK GRAPHICS LEVEL string-variable

String variable is set to one of "ma", "mb", "mc", "0a", "0b",
"0c", "1a", "1b", "1c", "2a", "2b", "2c".

INQUIRE LIST OF AVAILABLE GDP

   gks-statement     > ASK GDP LIST wkstn-type-select
                       numeric-variable-vector

INQUIRE LIST OF AVAILABLE WORKSTAION TYPES

   gks-statement     > ASK DEVICE LIST string-variable-vector
```

INQUIRE LIST OF COLOUR INDICES

```
gks-statement     > ASK COLOR LIST wkstn?
                    numeric-variable-vector
```

INQUIRE LIST OF NORMALISATION TRANSFORMATION NUMBERS

```
gks-statement     > ASK TRANSFORMATION LIST
                    numeric-variable-vector
```

INQUIRE MAXIMUM DISPLAY SURFACE SIZE (See Section 13)

```
gks-statement     > ASK DEVICE SIZE wkstn-type-select?
                    numeric-variable comma numeric-variable
                    comma string-variable
```

INQUIRE MAXIMUM NORMALISATION TRANSFORMATION NUMBER

```
gks-statement     > ASK MAX TRANSFORMATION numeric-variable
```

INQUIRE NORMALISATION TRANSFORMATION (See Section 13)

```
gks-statement     > ASK (WINDOW / VIEWPORT) tran-number?
                    boundary-variables
```

INQUIRE OPERATING STATE VALUE

```
gks-statement     > ASK GRAPHICS STATE string-variable
```

INQUIRE PATTERN FACILITIES

```
gks-statement     > ASK PATTERN TYPES wkstn-type-select?
                    numeric-variable
```

INQUIRE PIXEL (See Section 13)

```
gks-statement     > ASK PIXEL VALUE wkstn? point-location
                    numeric-variable
```

INQUIRE PIXEL ARRAY (See Section 13)

```
gks-statement     > ASK PIXEL ARRAY wkstn? point-location
                    numeric-array (comma string-variable)?
```

INQUIRE PIXEL ARRAY DIMENSIONS (See Section 13)

```
gks-statement    > ASK PIXEL SIZE wkstn? left-parenthesis
                   point-pair right-parenthesis
                   numeric-variable comma numeric-variable
```

INQUIRE POLYLINE FACILITIES

```
gks-statement    > ASK LINE TYPES wkstn-type-select?
                   numeric-variable-vector
                   comma numeric-variable comma
                   numeric-variable
                   comma numeric-variable comma
                   numeric-variable comma
                   numeric-variable
```

INQUIRE POLYMARKER FACILITIES

```
gks-statement    > ASK POINT TYPES wkstn-type-select?
                   numeric-variable-vector
                   comma numeric-variable comma
                   numeric-variable
                   comma numeric-variable comma
                   numeric-variable comma
                   numeric-variable
```

INQUIRE PREDEFINED COLOUR REPRESENTATION

```
gks-statement    > ASK PREDEFINED COLOR MIX type-index?
                   mix-list
type-index       = left-parenthesis wkstn-type comma index
                   right-parenthesis
```

INQUIRE PREDEFINED FILL AREA REPRESENTATION

```
gks-statement    > ASK PREDEFINED AREA TYPES type-index?
                   string-variable comma
                   numeric-variable comma numeric-variable
```

INQUIRE PREDEFINED PATTERN REPRESENTATION

```
gks-statement    > ASK PREDEFINED PATTERN type-index?
                   numeric-variable-matrix
```

INQUIRE PREDEFINED POLYLINE REPRESENTATION

```
gks-statement     > ASK PREDEFINED LINE TYPES type-index?
                    numeric-variable comma numeric-variable
                    comma numeric-variable
```

INQUIRE PREDEFINED POLYMARKER REPRESENTATION

```
gks-statement     > ASK PREDEFINED POINT TYPES type-index?
                    numeric-variable comma numeric-variable
                    comma numeric-variable
```

INQUIRE PREDEFINED TEXT REPRESENTATION

```
gks-statement     > ASK PREDEFINED TEXT type-index?
                    numeric-variable comma string-variable
                    comma numeric-variable comma
                    numeric-variable comma numeric-variable
```

INQUIRE SET OF OPEN WORKSTATIONS

```
gks-statement     > ASK DEVICE OPEN LIST
                    numeric-variable-vector
```

INQUIRE TEXT EXTENT

```
gks-statement     > ASK TEXT SIZE wkstn? left-parenthesis
                    coordinate-pair comma string-expression
                    right-parenthesis point-location
                    comma numeric-variable-matrix
```

INQUIRE TEXT FACILITIES

```
gks-statement     > ASK TEXT TYPES wkstn-type-select
                    numeric-variable-vector comma
                    string-variable-vector /
                    ASK TEXT HEIGHT RANGE range /
                    ASK TEXT EXPAND RANGE range
string-variable-vector = string-array left-parenthesis
                         question-mark right-parenthesis
range             = numeric-variable comma numeric-variable
                    comma numeric-variable
```

INQUIRE WORKSTATION CATEGORY

```
gks-statement     > ASK DEVICE CATEGORY wkstn-type-select?
                    string-variable
```

INQUIRE WORKSTATION CLASSIFICATION

```
gks-statement    > ASK DEVICE CLASS wkstn-type-select?
                   string-variable
```

INQUIRE WORKSTATION CONNECTION AND TYPE

```
gks-statement    > ASK DEVICE CONNECTION wkstn?
                   string-variable comma numeric-variable
```

INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES

```
gks-statement    > ASK DEVICE UPDATE wkstn? string-variable
                   comma string-variable comma string-variable
                   comma string-variable
```

INQUIRE WORKSTATION STATE

```
gks-statement    > ASK DEVICE STATE wkstn? string-variable
```

INQUIRE WORKSTATION TRANSFORMATION (See Section 13)

```
gks-statement    > ASK DEVICE (WINDOW / VIEWPORT) wkstn?
                   boundary-variables
```

F8. Initializing Input Functions

INITIALISE CHOICE (See also Section 13)

```
gks-statement    > SET CHOICE wkstn? device-select AT
                   numeric-expression PROMPT TYPE
                   numeric-expression
                   ECHO AREA boundaries RECORD data-record
```

INITIALISE LOCATOR (See also Section 13)

```
gks-statement    > SET POINT wkstn? device-select AT
                   coordinate-pair TRAN numeric-expression
                   PROMPT TYPE numeric-expression
                   ECHO AREA boundaries RECORD data-record
```

INITIALISE STRING

```
gks-statement    > SET STRING wkstn? device-select AT
                   string-expression
                   PROMPT TYPE numeric-expression
                   ECHO AREA boundaries RECORD data-record
```

INITIALISE STROKE (See also Section 13)

```
gks-statement     > SET MULTIPOINT wkstn? device-select AT
                    array-point-list TRAN numeric-expression
                    PROMPT TYPE numeric-expression
                    ECHO AREA boundaries RECORD data-record
```

INITIALISE VALUATOR (See also Section 13)

```
gks-statement     > SET VALUE wkstn? device-select AT
                    numeric-expression
                    PROMPT TYPE numeric-expression
                    ECHO AREA boundaries RECORD data-record
```

F9. Enquiry Input Functions

INQUIRE CHOICE DEVICE STATE

```
gks-statement     > ASK CHOICE STATE wkstn? device-select
                    choice-state (comma choice-state)*
choice-state      = basic-device-state /
                    INITIAL numeric-variable
basic-device-state = MODE string-variable /
                    ECHO STATE string-variable /
                    PROMPT TYPE numeric-variable /
                    ECHO AREA numeric-variable comma
                    numeric-variable comma numeric-variable
                    comma numeric-variable /
                    RECORD data-record
```

INQUIRE DEFAULT CHOICE DEVICE DATA

```
gks-statement     > ASK CHOICE DEFAULTS wkstn-device-select
                    choice-default (comma choice-default)*
wkstn-device-select = left-parenthesis wkstn-type comma
                      device right-parenthesis
device            = index
choice-default    = basic-default / INITIAL numeric-variable
basic-default     = PROMPT TYPE numeric-variable-array /
                    ECHO AREA numeric-variable comma
                    numeric-variable comma numeric-variable
                    comma numeric-variable /
                    RECORD data-record
```

```
INQUIRE DEFAULT LOCATOR DEVICE DATA

   gks-statement     > ASK POINT DEFAULTS wkstn-device-select
                       point-default (comma point-default)*
   point-default     = basic-default / INITIAL
                       coordinate-variables

INQUIRE DEFAULT STRING DEVICE DATA

   gks-statement     > ASK STRING DEFAULTS wkstn-device-select
                       string-default (comma string-default)*
   string-default    = basic-default / INITIAL string-variable

INQUIRE DEFAULT STROKE DEVICE DATA

   gks-statement     > ASK MULTIPOINT DEFAULTS wkstn-device-select
                       basic-default (comma basic-default)*

INQUIRE DEFAULT VALUATOR DEVICE DATA

   gks-statement     > ASK VALUE DEFAULTS wkstn-device-select
                       value-default (comma value-default)*
   value-default     = basic-default / INITIAL numeric-variable

INQUIRE LOCATOR DEVICE STATE

   gks-statement     > ASK POINT STATE wkstn? device-select
                       point-state (comma point-state)*
   point-state       = basic-device-state /
                       INITIAL coordinate-variables

INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES (See Sec. 13)

   gks-statement     > ASK MAX CHOICE DEVICE wkstn-type-select?
                       numeric-variable /
                       ASK MAX STRING DEVICE wkstn-type-select?
                       numeric-variable /
                       ASK MAX POINT DEVICE wkstn-type-select?
                       numeric-variable /
                       ASK MAX MULTIPOINT DEVICE wkstn-type-select?
                       numeric-variable /
                       ASK MAX VALUE DEVICE wkstn-type-select?
                       numeric-variable
```

INQUIRE STRING DEVICE STATE

```
gks-statement    > ASK STRING STATE wkstn? device-select
                   string-state (comma string-state)*
string-state     = basic-device-state /
                   INITIAL string-variable
```

INQUIRE STROKE DEVICE STATE

```
gks-statement    > ASK MULTIPOINT STATE wkstn? device-select
                   multi-state (comma multi-state)*
multi-state      = basic-device-state /
                   INITIAL numeric-variable-vector comma
                   numeric-variable-vector
```

INQUIRE VALUATOR DEVICE STATE

```
gks-statement    > ASK VALUE STATE wkstn? device-select
                   value-state (comma value-state)*
value-state      = basic-device-state /
                   INITIAL numeric-variable
```

F10. Request Mode Input Functions

REQUEST CHOICE (See Section 13)

```
gks-statement    > LOCATE CHOICE wkstn? device-select?
                   start-value? colon numeric-variable
```

REQUEST LOCATOR (See Section 13)

```
gks-statement    > LOCATE POINT wkstn? device-select?
                   initial-point? colon coordinate-variables
                   (TRAN numeric-variable)?
```

REQUEST STRING

```
gks-statement    > LOCATE STRING wkstn? device-select?
                   colon string-variable
```

REQUEST STROKE (See Section 13)

```
gks-statement    > MAT LOCATE POINT wkstn? device-select?
                   initial-point? colon array-locate-object
                   (TRAN numeric-variable)?
```

REQUEST VALUATOR (See Section 13)

gks-statement > LOCATE VALUE wkstn? device-select? range-select? start-value? colon numeric-variable

F11. Set Mode Input Functions

SET CHOICE MODE

gks-statement > SET CHOICE MODE wkstn? device-select string-expression comma string-expression

SET LOCATOR MODE

gks-statement > SET POINT MODE wkstn? device-select string-expression comma string-expression

SET STRING MODE

gks-statement > SET TEXT MODE wkstn? device-select string-expression comma string-expression

SET STROKE MODE

gks-statement > SET MULTIPOINT MODE wkstn? device-select string-expression comma string-expression

SET VALUATOR MODE

gks-statement > SET VALUE MODE wkstn? device-select string-expression comma string-expression

Appendix G.
Differences between Minimal BASIC and BASIC

The differences between Minimal BASIC and core BASIC may be classified as either syntactic incompatibilities or semantic (run-time) differences.

G1 Syntactic Differences. With the following exception, the core module forms an upward compatible syntactic extension of American National Standard Minimal BASIC, ANSI X3.60-1978.

(1) All arrays in a standard conforming program must be dimensioned before use.

Programs written in Minimal BASIC may therefore produce errors when run on an implementation that conforms to the core. Such programs may be modified to run correctly as follows:

(1) Identify all arrays which are implicitly dimensioned.
(2) Insert a dimension-statement covering each such array with upper bound equal to 10. Each such dimension-statement must follow an option-base-statement, if any, and precede any reference to the arrays contained in the dimension-statement.

For example, if a vector A is used in a Minimal BASIC program but is not dimensioned there, inserting

```
DIM A(10)
```

will cause the program to run correctly with respect to the vector A. Since array-names in Minimal BASIC are limited to single letters, there can be no more than 26 such changes needed.

G2 Semantic Differences. In addition, the core module differs from Minimal BASIC in several other ways that may be classified as "run-time." As a result, a Minimal BASIC program run under a BASIC implementation might produce slightly different results.

(1) The default lower bound for arrays is 1, not 0 as in Minimal Basic.

Programs in Minimal Basic can be made to run correctly if the following statement is introduced prior to any DIM statement:

```
OPTION BASE 0
```

(2) The core module specifies that arithmetic be carried out using a floating-point decimal representation, with at least ten decimal digits of precision, whereas Minimal BASIC is more permissive in allowing arithmetic to be carried out using other representations (e.g., floating-point binary), with at least six decimal digits of precision (cf. 5.6). The only effect should be that the program gives more precise results, which should not cause problems for the user. An option is provided that permits NATIVE arithmetic, which might be defined as in Minimal BASIC for a given implementation.

(3) The default maximum length for strings must be at least 132, not 18 as in Minimal BASIC. The only difference is that a program might not get a string-overflow exception, which it would have gotten in Minimal BASIC. The old behavior can be restored by declaring the maximum length of the strings to be the old maximum.

(4) It is not necessary to prevalidate an entire input-reply before assignment of values to variables takes place, whereas this was required in Minimal BASIC. Thus, an input-reply of "2,4,x" in response to INPUT I, A(I), J could change the value of A(2), whereas this is not allowed in Minimal BASIC.

(5) Certain exceptions -- overflow, division by zero, and raising to a negative power -- are fatal exceptions in BASIC and nonfatal in Minimal BASIC. However, since ANSI X3.60-1978 specifies that nonfatal exceptions can be treated as fatal under certain circumstances, a Minimal BASIC program should not rely on these exceptions being nonfatal.

Appendix H.
Language Elements under Consideration for Future Removal

The gosub-statement, on-gosub-statement, and the return-statement are under consideration for future removal. It is recommended that as users write new programs, or maintain existing programs, they refrain from using these statements, in order to improve compatibility with future versions of this standard.

The GOSUB facility is being considered for removal because it encourages poor programming practice by allowing the construction of subroutines with several entry points. Furthermore, these "subroutines" are not delineated by any distinctive syntax; any line of a program may be the beginning of such a subroutine. Users are encouraged to avail themselves of the subprogram facilities (see 9.2) described in this standard when they need subroutines.

Furthermore, the GOSUB facility interacts in a complex way with other aspects of the language (e.g., internal-proc-defs, exception-handlers), thus making it more difficult to understand source code, to implement conforming language processors, and to describe the language correctly. Thus, programmers, implementors, teachers, and writers are all impeded in their work with BASIC.

X3.115-1984 Unformatted 80 Megabyte Trident Pack for Use at 370 tpi and 6000 bpi (General, Physical, and Magnetic Characteristics)
X3.116-1986 Recorded Magnetic Tape Cartridge, 4-Track, Serial 0.250 Inch (6.30 mm) 6400 bpi (252 bpmm), Inverted Modified Frequency Modulation Encoded
X3.117-1984 Printable/Image Areas for Text and Facsimile Communication Equipment
X3.118-1984 Financial Services — Personal Identification Number — PIN Pad
X3.119-1984 Contact Start/Stop Storage Disk, 158361 Flux Transitions per Track, 8.268 Inch (210 mm) Outer Diameter and 3.937 inch (100 mm) Inner Diameter
X3.120-1984 Contact Start/Stop Storage Disk
X3.121-1984 Two-Sided, Unformatted, 8-Inch (200-mm), 48-tpi, Double-Density, Flexible Disk Cartridge for 13 262 ftpr Two-Headed Application
X3.122-1986 Computer Graphics Metafile for the Storage and Transfer of Picture Description Information
X3.124-1985 Graphical Kernel System (GKS) Functional Description
X3.124.1-1985 Graphical Kernel System (GKS) FORTRAN Binding
X3.125-1985 Two-Sided, Double-Density, Unformatted 5.25-inch (130-mm), 48-tpi (1,9-tpmm), Flexible Disk Cartridge for 7958 bpr Use
X3.126-1986 One- or Two-Sided Double-Density Unformatted 5.25-inch (130-mm), 96 Tracks per Inch, Flexible Disk Cartridge
X3.127-1987 Unrecorded Magnetic Tape Cartridge for Information Interchange
X3.128-1986 Contact Start-Stop Storage Disk — 83 000 Flux Transitions per Track, 130-mm (5.118-in) Outer Diameter and 40-mm (1.575-in) Inner Diameter
X3.129-1986 Intelligent Peripheral Interface, Physical Level
X3.130-1986 Intelligent Peripheral Interface, Logical Device Specific Command Sets for Magnetic Disk Drive
X3.131-1986 Small Computer Systems Interface
X3.132-1987 Intelligent Peripheral Interface — Logical Device Generic Command Set for Optical and Magnetic Disks
X3.133-1986 Database Language —NDL
X3.135-1986 Database Language — SQL
X3.136-1986 Serial Recorded Magnetic Tape Cartridge for Information Interchange, Four and Nine Track
X3.139-1987 Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)
X3.140-1986 Open Systems Interconnection — Connection Oriented Transport Layer Protocol Specification
X3.141-1987 Data Communication Systems and Services — Measurement Methods for User-Oriented Performance Evaluation
X3.146-1987 Device Level Interface for Streaming Cartridge and Cassette Tape Drives
X3.147-1987 Intelligent Peripheral Interface — Logical Device Generic Command Set for Magnetic Tapes
X3.153-1987 Open Systems Interconnection — Basic Connection Oriented Session Protocol Specification
X3.156-1987 Nominal 8-Inch Rigid Disk Removable Cartridge
X3.157-1987 Recorded Magnetic Tape for Information Interchange, 3200 CPI
X3.158-1987 Serial Recorded Magnetic Tape Cassette for Information Interchange, 0.150 Inch (3.81 mm), 8000 bpi (315 bpmm), Group Code Recording.
X11.1-1977 Programming Language MUMPS
IEEE 416-1978 Abbreviated Test Language for All Systems (ATLAS)
IEEE 716-1982 Standard C/ATLAS Language
IEEE 717-1982 Standard C/ATLAS Syntax
IEEE 770X3.97-1983 Programming Language PASCAL
IEEE 771-1980 Guide to the Use of ATLAS
ISO 8211-1986 Specifications for a Data Descriptive File for Information Interchange
MIL-STD-1815A-1983 Reference Manual for the Ada Programming Language
NBS-ICST 1-1986 Fingerprint Identification — Data Format for Information Interchange

---

X3/TR1-82 Dictionary for Information Processing Systems (Technical Report)

American National Standards for Information Processing

X3.1-1987 Synchronous Signaling Rates for Data Transmission
X3.2-1970 Print Specifications for Magnetic Ink Character Recognition
X3.4-1986 Coded Character Sets — 7-Bit ASCII
X3.5-1970 Flowchart Symbols and Their Usage
X3.6-1965 Perforated Tape Code
X3.9-1978 Programming Language FORTRAN
X3.11-1969 General Purpose Paper Cards
X3.14-1983 Recorded Magnetic Tape (200 CPI, NRZI)
X3.15-1976 Bit Sequencing of the American National Standard Code for Information Interchange in Serial-by-Bit Data Transmission
X3.16-1976 Character Structure and Character Parity Sense for Serial-by-Bit Data Communication in the American National Standard Code for Information Interchange
X3.17-1981 Character Set for Optical Character Recognition (OCR-A)
X3.18-1974 One-Inch Perforated Paper Tape
X3.19-1974 Eleven-Sixteenths-Inch Perforated Paper Tape
X3.20-1967 Take-Up Reels for One-Inch Perforated Tape
X3.21-1967 Rectangular Holes in Twelve-Row Punched Cards
X3.22-1983 Recorded Magnetic Tape (800 CPI, NRZI)
X3.23-1985 Programming Language COBOL
X3.25-1976 Character Structure and Character Parity Sense for Parallel-by-Bit Data Communication in the American National Standard Code for Information Interchange
X3.26-1980 Hollerith Punched Card Code
X3.27-1987 Magnetic Tape Labels and File Structure
X3.28-1976 Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links
X3.29-1971 Specifications for Properties of Unpunched Oiled Paper Perforator Tape
X3.30-1986 Representation for Calendar Date and Ordinal Date
X3.31-1973 Structure for the Identification of the Counties of the United States
X3.32-1973 Graphic Representation of the Control Characters of American National Standard Code for Information Interchange
X3.34-1972 Interchange Rolls of Perforated Tape
X3.37-1987 Programming Language APT
X3.38-1972 Identification of States of the United States (including the District of Columbia)
X3.39-1986 Recorded Magnetic Tape (1600 CPI, PE)
X3.40-1983 Unrecorded Magnetic Tape (9-Track 800 CPI, NRZI; 1600 CPI, PE; and 6250 CPI, GCR)
X3.41-1974 Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Standard Code for Information Interchange
X3.42-1975 Representation of Numeric Values in Character Strings
X3.43-1986 Representations of Local Time of Day
X3.44-1974 Determination of the Performance of Data Communication Systems
X3.45-1982 Character Set for Handprinting
X3.46-1974 Unrecorded Magnetic Six-Disk Pack (General, Physical, and Magnetic Characteristics)
X3.47-1977 Structure for the Identification of Named Populated Places and Related Entities of the States of the United States for Information Interchange
X3.48-1986 Magnetic Tape Cassettes (3.81-mm [0.150-Inch] Tape at 32 bpmm [800 bpi], PE)
X3.49-1975 Character Set for Optical Character Recognition (OCR-B)
X3.50-1986 Representations for U.S. Customary, SI, and Other Units to Be Used in Systems with Limited Character Sets
X3.51-1986 Representations of Universal Time, Local Time Differentials, and United States Time Zone References
X3.52-1976 Unrecorded Single-Disk Cartridge (Front Loading, 2200 BPI) (General, Physical, and Magnetic Requirements)
X3.53-1976 Programming Language PL/I
X3.54-1986 Recorded Magnetic Tape (6250 CPI, Group Coded Recording)
X3.55-1982 Unrecorded Magnetic Tape Cartridge, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase encoded
X3.56-1986 Recorded Magnetic Tape Cartridge, 4 Track, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded
X3.57-1977 Structure for Formatting Message Headings Using the American National Standard Code for Information Interchange for Data Communication Systems Control
X3.58-1977 Unrecorded Eleven-Disk Pack (General, Physical, and Magnetic Requirements)
X3.60-1978 Programming Language Minimal BASIC
X3.61-1986 Representation of Geographic Point Locations
X3.62-1987 Paper Used in Optical Character Recognition (OCR) Systems
X3.63-1981 Unrecorded Twelve-Disk Pack (100 Megabytes) (General, Physical, and Magnetic Requirements)
X3.64-1979 Additional Controls for Use with American National Standard Code for Information Interchange
X3.66-1979 Advanced Data Communication Control Procedures (ADCCP)
X3.72-1981 Parallel Recorded Magnetic Tape Cartridge, 4 Track, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded
X3.73-1980 Single-Sided Unformatted Flexible Disk Cartridge (for 6631-BPR Use)
X3.74-1987 Programming Language PL/I, General-Purpose Subset
X3.76-1981 Unformatted Single-Disk Cartridge (Top Loading 200 tpi 4400 bpi) (General, Physical, and Magnetic Requirements)
X3.77-1980 Representation of Pocket Select Characters
X3.78-1981 Representation of Vertical Carriage Positioning Characters in Information Interchange
X3.79-1981 Determination of Performance of Data Communications Systems That Use Bit-Oriented Communication Procedures
X3.80-1981 Interfaces between Flexible Disk Cartridge Drives and Their Host Controllers
X3.82-1980 One-Sided Single-Density Unformatted 5.25-Inch Flexible Disk Cartridge (for 3979-BPR Use)
X3.83-1980 ANSI Sponsorship Procedures for ISO Registration According to ISO 2375
X3.84-1981 Unformatted Twelve-Disk Pack (200 Megabytes)(General, Physical, and Magnetic Requirements
X3.85-1981 1/2-Inch Magnetic Tape Interchange Using a Self Loading Cartridge
X3.86-1980 Optical Character Recognition (OCR) Inks
X3.88-1981 Computer Program Abstracts
X3.89-1981 Unrecorded Single-Disk, Double-Density Cartridge (Front Loading, 2200 bpi, 200 tpi) (General, Physical, and Magnetic Requirements)
X3.91M-1987 Storage Module Interfaces
X3.92-1981 Data Encryption Algorithm
X3.93M-1981 OCR Character Positioning
X3.94-1985 Programming Language PANCM
X3.95-1982 Microprocessors — Hexadecimal Input/Output, Using 5-Bit and 7-Bit Teleprinters
X3.96-1983 Continuous Business Forms (Single-Part)
X3.98-1983 Text Information Interchange in Page Image Format (PIF)
X3.99-1983 Print Quality Guideline for Optical Character Recognition (OCR)
X3.100-1983 Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment for Packet Mode Operation with Packet Switched Data Communications Network
X3.101-1984 Interfaces Between Rigid Disk Drive(s) and Host(s)
X3.102-1983 Data Communication Systems and Services — User-Oriented Performance Parameters
X3.103-1983 Unrecorded Magnetic Tape Minicassette for Information Interchange, Coplanar 3.81 mm (0.150 in)
X3.104-1983 Recorded Magnetic Tape Minicassette for Information Interchange, Coplanar 3.81 mm (0.150 in), Phase Encoded
X3.105-1983 Data Link Encryption
X3.106-1983 Modes of Operation for the Data Encryption Algorithm
X3.110-1983 Videotex/Teletext Presentation Level Protocol Syntax
X3.111-1986 Optical Character Recognition (OCR) Matrix Character Sets for OCR-M
X3.112-1984 14-in (356-mm) Diameter Low-Surface-Friction Magnetic Storage Disk
X3.113-1987 Programming Language FULL BASIC
X3.114-1984 Alphanumeric Machines; Coded Character Sets for Keyboard Arrangements in ANSI **X4**.23-1982 and **X4**.22-1983

(Continued on reverse)