INTERNATIONAL STANDARD

ISO/IEC 10279

First edition
1991-10-15

Information technology — Programming languages — Full BASIC

Technologies de l'information — Langages de programmation — Full BASIC

Reference number
ISO/IEC 10279:1991(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 10279 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Annexes A and B are for information only.

ISO/IEC Copyright Office • Case Postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

Information technology — Programming languages — Full BASIC

1 Scope

This International Standard specifies the programming language Full BASIC and is derived from the American National Standard X3.113-1987. For details of the syntax and semantics see ANSI X3.113-1987 which specifies

- the syntax of programs written in BASIC, including *core* BASIC and various extensions thereto;
- 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;
- 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;
- the semantic rules for interpreting the meaning of a program written in BASIC;
- errors and exceptional circumstances to be detected and also the manner in which such errors and exceptional circumstances are to be handled

This International Standard also refers to ECMA-116 for the specification of *mini-graphics*. Note: ECMA-116 is based on ANSI X3.113-1987.

This International Standard specifies its own conformance subsets, which include those specified in ANSI X3.113-1987 and ECMA-116.

This International Standard is designed to promote the interchangeability of BASIC programs among a variety of automatic data processing systems.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and of ISO maintain registers of currently valid International Standards.

ANSI X3.113-1987: *Information systems - programming languages - full BASIC*.

STANDARD ECMA-116: 1986, *BASIC*.

3 Technical content

3.1 Principal technical content

Apart from the conformance rules the technical content of this International Standard is defined in part by Sections 3 to 16 of the ANSI X3.113-1987 (which will be referred to a ANS BASIC). The definition of *reserved word* in ANS BASIC Section 3.2 is for information only with respect to this International Standard.

The technical content for the *mini-graphics* module of this International Standard is defined by Section 13 of ECMA-116.

3.2 Additional reserved words

For the *subset core,* insert the following paragraph after paragraph 6 of Subsection 4.4.2 of ANS BASIC.

"The keywords ACCESS, AND, ANGLE, AREA, ARITHMETIC, ASK, AT, BASE, BEGIN, BREAK, CALL, CASE, CHAIN, CLEAR, CLIP, CLOSE, COLLATE, COLOR, DATA, DATUM, DEBUG, DECIMAL, DECLARE, DEF, DEGREES, DEVICE, DIM, DISPLAY, DO, ELAPSED, ELSEIF, END, ERASE, ERASABLE, EXIT, EXTERNAL, FILETYPE, FOR, FUNCTION, GO, GOSUB, GOTO, GRAPH, IF, IMAGE, INPUT, INTERNAL, IS, LENGTH, LET, LINE, LINES, LOOP, MARGIN, MAT, MISSING, NAME, NATIVE, NEXT, NUMERIC, OFF, ON, OPEN, OPTION, OR, ORGANIZATION, OUTIN, OUTPUT, POINT, POINTER, POINTS, PROGRAM, PROMPT, RADIANS, RANDOMIZE, READ, RECSIZE, RECTYPE, REST, RESTORE, RETURN, SAME, SELECT, SEQUENTIAL, SET, SETTER, SIZE, SKIP, STANDARD, STATUS, STEP, STOP, STREAM, STRING, STYLE, SUB, TAB, TEXT, THEN, THERE, TIMEOUT, TO, TRACE, UNTIL, USING, VARIABLE, VIEWPORT, WHILE, WINDOW, WITH, WRITE, and ZONEWIDTH shall not be used as identifiers."

4 Conformance

There are two aspects of conformance to a set of modules in this International Standard: conformance by a program written in the BASIC language, and conformance by an implementation which processes such programs. 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 ANS BASIC Appendix C).

4.1 Modules

The programming language defined by this International Standard is organized in a modular fashion. Conformance to this International Standard is defined with respect to particular sets of the following fifteen modules and combinations thereof:

a) A *core* module, which encompasses all programs whose syntax conforms to ANS BASIC Sections 4 to 10, parts of ANS BASIC Section 11 (excluding internal-format record and native-format record files), and ANS BASIC Section 12.

b) A *subset core* module, which encompasses all programs whose syntax conforms to ANS BASIC Sections 4 to 10 (except that a substitute definition of *reserved word* applies -- see 3.2), parts of ANS BASIC Section 11 (excluding enhanced-internal and enhanced-native files), and ANS BASIC Subsection 12.2. (The *subset core* module is identical to the *core* module except that the list of reserved words is larger and exception handling is excluded.)

c) An *enhanced-internal file* module, which encompasses all programs whose syntax conforms to the enhanced production rules in ANS BASIC Section 11 (lacking the prefix "N"), together with the *core*.

d) An *enhanced-internal file subset* module, which encompasses all programs whose syntax conforms to the enhanced production rules in ANS BASIC Section 11 (lacking the prefix "N"), together with the *subset core*.

e) An *enhanced-native file* module, which encompasses all programs whose syntax conforms to the enhanced native production rules in ANS BASIC Section 11 (indicated with the prefix "N"), together with the *core*.

f) An *enhanced-native file subset* module, which encompasses all programs whose syntax conforms to the enhanced native production rules in ANS BASIC Section 11 (indicated with the prefix "N"), together with the *subset core*.

g) A *graphics* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 13, together with the *core*.

h) A *graphics subset* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 13, together with the *subset core*.

i) A *mini graphics* module, which encompasses all programs whose syntax conforms to ECMA-116 Section 13 (see Annex B), together with the *core*.

j) A *mini graphics subset* module, which encompasses all programs whose syntax conforms to ECMA-116 Section 13 (see Annex B), together with the *subset core*.

k) A *real-time* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 14, together with the *core*.

l) A *real-time subset* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 14, together with the *subset core*.

m) A *fixed decimal* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 15, together with the *core*.

n) A *fixed decimal subset* module, which encompasses all programs whose syntax conforms to ANS BASIC Section 15, together with the *subset core*.

o) An *editing* module, which encompasses all unsorted programs and editing commands whose syntax conforms to ANS BASIC Section 16.

In addition, the Conformance Modules shall include those described in ANS X3-113, Section 2, and in ECMA-116, Section 2. (Note: See Annex A for details on the relationship between these conformance modules and the conformance modules defined in ANS X3-113 and ECMA-116.)

4.2 Program conformance

A program conforms to a set of modules in this International Standard only when

- the program and each statement or other syntactic element contained therein is syntactically valid according to the syntactic rules specified by this International Standard as belonging to that set;

- the program as a whole violates none of the global constraints imposed by this International Standard on the application of the syntactic rules.

4.3 Implementation conformance

An implementation conforms to a set of modules in this International Standard only when

- it accepts and processes all programs conforming to that set of modules in this International Standard;
- it reports reasons for rejecting any program which does not conform to that set of modules in this International Standard;
- it interprets errors and exceptional circumstances according to the specifications of this International Standard;
- it interprets the semantics of each statement of a conforming program according to the specifications in this International Standard;
- it interprets the semantics of a conforming program as a whole according to the specifications in this International Standard;
- it accepts as input, manipulates, and can generate as output numbers of at least the precision and range specified in this International Standard;
- 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 International Standard;
- 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 International Standard;
- it is accompanied by documentation available to the user that describes and identifies all enhancements to the language defined in this International Standard.

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

In addition, an implementation conforms to the editing requirements of this International Standard if it accepts and processes unsorted programs and editing commands according to the specifications in ANS BASIC Section 16.

4.4 Errors

This International 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 International 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.

4.5 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 International Standard cannot be followed or that some resource constraint is about to be exceeded. All exceptions described in this International Standard shall be detected, reported, and processed when they occur, unless some mechanism provided in ANS BASIC Subsection 12.1 or in an enhancement to this International 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 International Standard. If no recovery procedure is specified in this International

Standard, or if restrictions imposed by the hardware or the operating environment make it impossible to follow the procedure specified in this International 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 ANS BASIC Subsection 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 International 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.

Annex A
(informative)

Relationship of this International Standard to ANSI and ECMA standards

It is the intention of this International Standard that all programs and implementations that conform to ANS BASIC, X3.113-1987, or to ECMA BASIC, ECMA-116, shall automatically conform to ISO/IEC BASIC.

In addition, this International Standard allows other conformance modules. Subclause 4.1 of this International Standard defines fifteen conformance modules. These fifteen modules are based on nine portions of the ANSI and ECMA Standards. The content of these fifteen modules in terms of the nine portions is shown below.

In particular, the core and six optional modules specified in ANS BASIC correspond to modules a), c), e), g), k), m), and o). The four conformance modules of ECMA-116 are shown in the following the table A.1. The ECMA conformance modules are

ECMA BASIC-1	(ECMA-1)
ECMA-1 with Mini-Graphics	(E-1+MG)
ECMA BASIC-2	(ECMA-2)
ECMA-2 with Mini-Graphics	(E-2+MG)

Module	Core	Subset Core	Files Internal	Files Native	Graph-ics	Mini-Graph.	Real-Time	Fixed Decimal	Editing
a)	X								
b)		X							
c)	X		X						
d)		X	X						
e)	X			X					
f)		X		X					
g)	X				X				
h)		X			X				
i)	X					X			
j)		X				X			
k)	X						X		
l)		X					X		
m)	X							X	
n)		X						X	
o)	X								X
ECMA-1		X							
E-1+MG		X				X			
ECMA-2	X		X	X				X	
E-2+MG	X		X	X		X		X	

Annex B
(informative)

The Mini Graphics Module

This annex reproduces (except for typographical corrections) Section 13 in ECMA-116 --*mini-graphics* , which can be described informally as consisting of ANS BASIC Subsections 13.1 through 13.3 with the following deletions:

From 13.1. None.

From 13.2. Delete references and production rules associated with more than a single TEXT style, size, or orientation. Delete references to COLOR MIX. Reduce the minimum number of available line styles from four to three. Reduce the minimum number of available point styles from five to three. Delete exceptions 11073, 4102, and 11088.

From 13.3. Delete the facility for generating an array of color cells. Delete refer-ences to array (MAT) graphic output statements. Delete the second paragraph of 13.3.4.1 that deals with the interaction between ordinary output and graphics output statements. Delete all references to the height, justification, and orientation of text. Delete exceptions 6401 and 6402. Delete the reference in exception 11085 to the array-cells-statement.

13. GRAPHICS

The facilities provided in section 13.1 through 13.3 are a subset of those provided by level 0b of the Graphical Kernel System (GKS) as defined in ISO 7942. The values of the EXTYPE function for exceptions defined in GKS are 11000 plus the value of the GKS error number.

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 WORSTATION #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 which 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, Unit$
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 must 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 (see Section 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 upper-case and lower-case. 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 keyword 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 widow 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.

The figure below illustrates the relationship between the window, the viewport, the device window, and the device viewport; clipping is assumed "ON".

[See ANSI X3.113-1987 or ECMA-116 for the figure.]

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.

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 of 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 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 as defined above. 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, respectively, of the GKS function INQUIRE WORKSTATION TRANSFORMATION with a workstation identifier of #0.

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.

13.2 Attributes and Screen Control

13.2.1 General Description

A graphical 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 and/or fill areas 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.

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.

This Standard provides text of one style and size that shall be output horizontally with the initial-point at the left.

13.2.2 Syntax

1.	imperative-statement	>	clear-statement
2.	clear-statement	=	CLEAR
3.	set-object	>	primitive-1 STYLE index / primitive-2 COLOR index
4.	primitive-2	=	primitive-1 / TEXT / AREA
5.	primitive-1	=	POINT / LINE
6.	rgb-list	>	[deleted]
7.	ask-object	>	primitive-1 STYLE numeric-variable / primitive-2 COLOR numeric-variable / MAX primitive-1 STYLE numeric-variable / MAX COLOR numeric-variable
8.	mix-list	=	[deleted]
9.	text-facet	=	[deleted]

13.2.3 Examples

3. LINE STYLE 2
   TEXT COLOR 5

7. 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 three. A line style of one must correspond to drawing of solid lines. A line style of two shall correspond to drawing of dashed lines. A line style of three shall correspond to 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 three. A point style of one must correspond to a dot (.), a point style of two to a plus sign (+), a point style of three to an asterisk (*). 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 the 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. The number of colors available is implementation-defined.

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 an ask-statement with the keywords MAX COLOR shall assign to the numeric-variable the largest distinct value available as an index for 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.

13.2.5 Exceptions

A color index in a set-statement with the keywords 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).

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.

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.

The following ask-objects correspond to various parameters of the GKS function INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES: LINESTYLE is linetype, POINSTYLE 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.

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. The effect of the graphic output statements depends on the current values of the various set-objects described in section 13.1 and 13.2.

13.3.2 Syntax

1.	imperative-statement	>	graphic-output-statement
2.	graphic-output-statement	>	geometric-statement / graphic-text-statement
3.	geometric-statement	>	graphic-verb geometric-object colon point-list
4.	graphic-verb	>	GRAPH
5.	geometric-objec	=	POINTS / LINES / AREA

6.	point-list	=	coordinate-pair (semicolon coordinate-pair)*
7.	coordinate-pair	=	numeric-expression comma numeric-expression
8.	array-geometric-statement	=	[deleted]
9.	size-select	=	[deleted]
10.	array-point-list	=	[deleted]
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	=	[deleted]
14.	point-pair	=	[deleted]

A graphic-output-statement with LINES as the geometric-object must contain at least two coordinate-pairs in its point-list. A graphic-output-statement with AREA as the geometric-object must contain at least three coordinate-pairs in its point-list.

13.3.3 Examples

```
3.    GRAPH LINES: 3,4; 5,6; 66.66,77.77
11.   GRAPH TEXT, AT XP, YP: "here is the label: " & TEXT$
      GRAPH TEXT, AT O,Y_VALUE, USING "##. ##^^^^": Y_VALUE
```

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 is 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.

13.3.4.2 The geometric-statement. The geometric-statement makes use of a sequence of points specified in problem coordinates. 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.

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.

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 used for labels shall have an implementation-defined size and style. The effect of clipping on characters which lie partly in and partly out of the viewport on the screen is implementation-defined.

13.3.5 Exceptions

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).

13.3.6 Remarks

The graphic-text-statement is designed to give easy access to a device's hardware-generated character set.

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.

GRAPH POINTS corresponds to the GKS function POLYMARKER. GRAPH LINES corresponds to the GKS function POLYLINE. GRAPH AREA corresponds 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.