Application Program

H20-0183-0

1620 Drafting System (1620-CX-04X) Programmer's Manual

This program can be used by any industry to produce a drawing as computer output. The program is designed particularly to produce mechanical detail drawings from language statement input.

This manual explains the language statements used to produce drawings. The manual presents the rules of the language, the various classes of statements, and examples of Drafting System programs and drawings.

CONTENTS

2.00 INTRODUCTION

The purpose of the 1620 Drafting System is to produce drawings. The desired drawing is described using the language of the system; the language is processed using the programs of the system; and a drawing is produced that consists of lines, arcs, and alphabetic text.

A draftsman produces a drawing, such as the one shown in Figure 1, by using traditional tools — scales, templates, and compasses. The 1620 Drafting System replaces these tools with a drafting language. The following statements are the "tools" which generate the drawing in Figure 1.

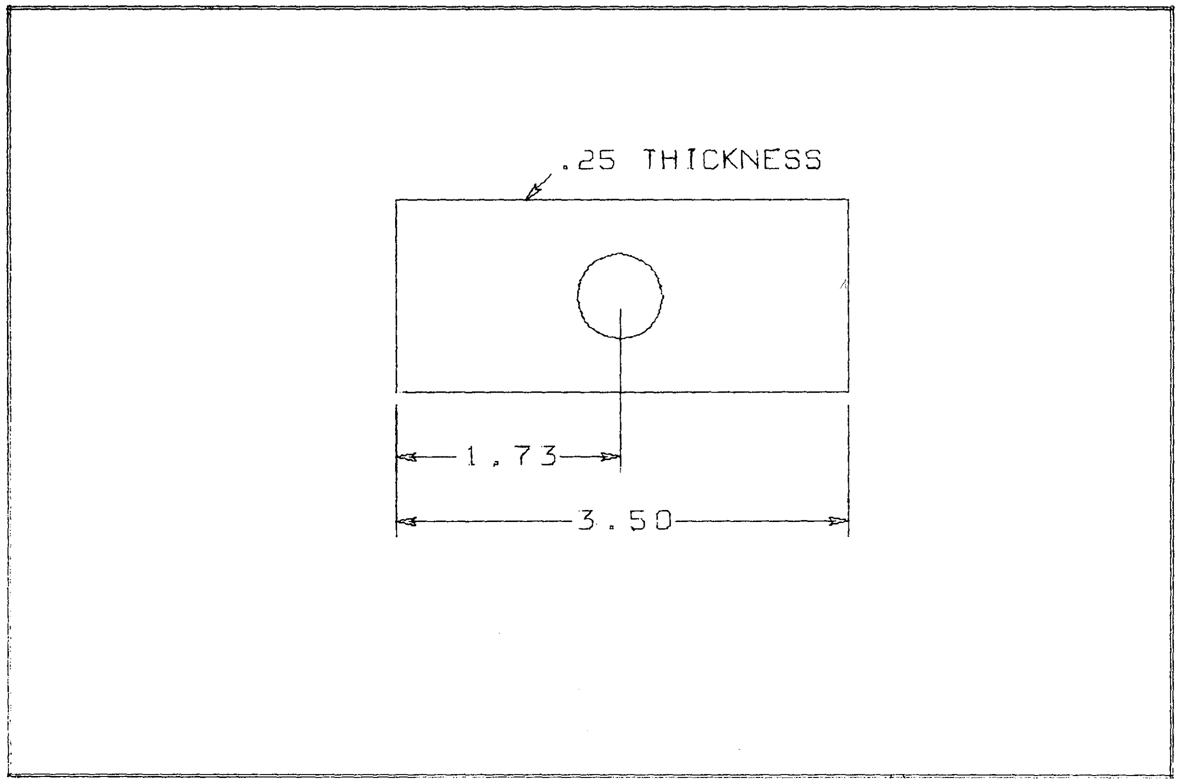


Figure 1

```
FRONT = VIEW/
     L1 = LINE/DX, 3.5
          LINE/DY, 16
          LINE/PPP, 0, 1.6
          LINE/0, 1.6, 0, 0
     C1 = CIRCLE/1.73, .8, 1/4
END/FRONT
ORIGIN/5, 5
DRAW/FRONT
DIMST/YSMALL, XCOMP, L1
     DIM/0, 0, POINT/CENTER, C1
     DIM/L1
NOTE/1, 1.6, .5, .5, @.25 THICKNESS@
FINI/
```

An examination of these statements shows that they consist of certain words (some familiar and some new), special symbols and punctuation, and an overall structure or grammar. The purpose of this manual is to explain each aspect of the language and how the language is used to produce drawings.

2.00.01 CONTENTS OF THE MANUAL

This manual is divided into the following sections, each of which assumes that the reader has a knowledge of the material in the preceding sections.

1. Format of the Language. A drawing is produced by writing language statements. The general rules for constructing statements are given in this section. Explanations are given for the various words and punctuation needed to write language statements.

2. Geometric Statements. The drawing of an object consists of lines and arcs. Geometric statements are used to describe these elements. The use of each statement to produce a desired geometric configuration is explained in this section.

3. Arithmetic Statements. These statements are used to direct the 1620 Drafting System to do computations. The quotient of two numbers and the sine of an angle are examples of the computations which the system can perform. All of the arithmetic features of the language are explained in this section.

4. Drawing Statements. Geometric statements describe the object to be drawn. Drawing statements tell how to draw the picture, and also what annotations to add to the picture. The statements used to scale an object and to place it on the paper are discussed in this section. Discussed, also, are the statements that produce notes and dimension lines.

5. Control Statements. After geometric and drawing statements are written and punched into cards, they must be processed by the 1620 Drafting System. Control statements are used to direct the processing operation. Included in this section, for example, are the statements to stop the computer to change the plotter's pen, or to notify the processor that a set of statements (a macro) is to be stored away for future use.

6. Examples of Programming. Presented in this section are sample drawings and the language statements used to create them. Shown, also, is the way to combine the various statements into a program that will produce a drawing.

7. Diagnostics.

8. Dictionary.

2.00.02 GENERAL PROCEDURES FOR PRODUCING A DRAWING

This is the general procedure for producing a drawing with the drafting system:

1. Write the necessary program statements for the drawing. These fall into two general categories — the geometric statements that describe the object to be drawn, and the drawing statements that tell how to draw the object and what kind of annotations to add to the picture.

2. Convert the statements into punched cards. This standard data processing operation converts the statements to a form that can be accepted by a computer.

3. Process the cards on the computer, using the 1620 Drafting System programs. The detailed procedures for operating the system on the computer are contained in the Operator's Manual.

4. Remove the drawing from the 1627 Plotter.

2.00.03 GENERAL DESCRIPTION OF THE PROCESSORS

The 1620 Drafting System is composed of three main processors: the compiler, the part processor, and the drawing processor. Detailed descriptions of these processors are contained in the System Manual.

The purpose of the compiler is to read the drafting statements and convert them to 1620 object instructions. Compilation requires two steps: first, a model that represents the input statement is constructed in the computer; second, the model is converted to object code. The complete object code program is written onto the file for execution by the next phase.

In the part processor phase, the object code program is read and executed. All necessary arithmetic and geometric calculations are performed by subroutines during this phase. Control of the execution of these subroutines is shared jointly by the part processor supervisor and the object program itself. The purpose of this phase is to produce two files of output records. The first file is composed of lines, circles, and arcs which were described in geometric-definition statements; this set of records is called the part model file. Drawing statements cause the second file to be written. This set of records is called the drawing command file; it contains the translated form of the drawing commands that appear in the input statements.

The drawing processor is called into core as soon as the second phase is finished. The first thing that happens in this phase is the processing of the drawing commands. Whenever one of these commands references the part model file — for example, a DRAW statement — the drawing processor begins to read the part model and to convert it into lines on the 1627 Plotter. Commands to crosshatch or dimension a drawing require that the drawing processor write out an intermediate file of records and then read this file back into core to initiate drawing.

2.01 FORMAT OF THE LANGUAGE

The complete set of words written to produce a drawing in the 1620 Drafting System is called a program. A program consists of statements. Each statement is written for a specific purpose. Some statements describe the geometric content of the object to be drawn, such as lines and circles; some cause calculations, such as the addition of several numbers; some cause drawing of notes and dimensions; others control the overall action of the processor. Whatever its purpose, a statement consists of words and punctuation.

2.01.01 LANGUAGE WORDS

These words constitute the dictionary of the language. Each word has a specific use and must always be spelled exactly as shown in this manual. There are three ways of classifying language words (the divisions are arbitrary, but they aid in explaining the language):

1. Major and nonmajor words. Almost every statement begins with a language word. The words indicate the purpose of the statement. The word LINE is used to start a statement that describes a line; the word NOTE is used to start a statement that places a note on a drawing. A word of this type is called a major word. Major words are recognized easily, because they are always followed in a statement by a slash (/). Using this scheme of classification, all words in the language are either major words or nonmajor words.

2. Functional words. Language words may also be classified according to function. The table of contents for this manual reflects this kind of classification. The words VIEW and SHAPE are used in language statements that accomplish geometric grouping. POINT, LINE, CIRCLE, and ARC are words used for geometric definitions. Arithmetic functions are represented by language words, such as SIND, COSD, and SQRT. Statements which affect the drawing reference system contain the words SCALE and ORIGIN, so that these two words may be classified as reference-system words.

3. Modal words. By the first classification, the word SCALE is a major word. In the second scheme the word SCALE becomes a reference-system word. In the third way of classifying language words, the word SCALE is called a modal word. The word SCALE is used whenever the programmer wishes to produce a scaled drawing. For example, the statement SCALE/2 notifies the drafting system that a drawing of double scale is desired. The actual drawing is not done until a DRAW statement is given — something like DRAW/FRONT. During the process of drawing, the scale factor is applied continually. Any succeeding draw statements — DRAW/SIDE, for example — are also affected by the double scale. A word which has this kind of continuing effect on the system is called a modal word. The double scale factor, for instance, would remain in effect for all DRAW statements until another SCALE statement is given.

2.01.02 USER-CREATED WORDS

Language words, by themselves, do not produce a drawing. The program statements for a drawing consist of language words, together with words created by the programmer. For example, the language word POINT is part of the statement written by the programmer to define a point. One of the allowable ways to define a point in the language is to give the coordinates of the desired point. A statement of this form might be: POINT/4, 5. The numbers 4 and 5 are the coordinates of the point. These numbers are user-created words. Besides numbers, the user can also create literals and labels.

1. Numbers perform many functions in the language statements. Besides indicating the coordinates of a point, as shown above, numbers may also tell the length of a line, the radius of a circle, the size of lettering, and other appropriate values. In creating a number, the user must follow these rules:

   - Numbers consist of the digits 0 through 9 and a decimal point, if required.

   - Numbers can be up to eight significant digits in length.

   - The arithmetic sign of a number can be indicated with a plus (+) or a minus (-) sign. When there is no sign, a positive value is assumed.

Some valid numbers are:

```
 1.734
+65
 0.0
 0
 -.048
```

Some invalid numbers are:

```
23,000.04
12345678.99
```

The first invalid number contains a comma; this violates the first rule. The next invalid number contains more than eight digits; this violates the second rule.

2. Literals have several uses in the language. The most frequent use is in drawing annotations. General notes are placed on a drawing by writing TITLE statements. Such a statement might be: TITLE/2, 4, @PART NO. 478-BC@. The major word TITLE indicates the purpose of the statement; the numbers 2 and 4 represent the coordinates of the point at which the note is to be placed; the note to be drawn is shown, literally, in the statement by the information contained between the two @-signs. The following rules govern the creation of literals:

- Literals consist of alphameric characters (the letters A through Z, the digits 0 through 9), and the punctuation characters which are explained in section 2.01.03 of this manual.
- Literals can be up to 48 characters in length.
- Literals are always enclosed by @-signs. (The @-signs are not part of the literal and therefore do not affect the 48-character maximum length.)

Some valid literals are:

```
@JOHN JONES@
@SCALE IS 2@
@*@
@CHANGE PEN@
```

Some invalid literals are:

```
@WATCH TOLERANCE!@
@ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ@
@THREE@.25 RADIUS@
```

The first example is invalid because it contains an exclamation point. The second exceeds the maximum length for a literal. The third attempts to place an @-sign within a literal, violating the rule that a literal begins and ends with an @-sign.

3. Labels, or arbitrary names, can be assigned to geometric definitions, geometric groups, literals, or numbers that are to be referenced in later statements of a program. The rules for creating a label are:

   - The label must always begin with a letter (A-Z).
   - After the first letter, a label can consist of other letters and any of the digits (0-9).
   - A label can be one to six characters in length.
   - A label must not be identical to any drafting system language word.

   Some valid labels are:

```
SIDE
FRAME
P26
```

   Some invalid labels are:

```
26P
SIDE-A
BUSHING
LINE
```

   The first example is invalid because it begins with a number instead of a letter. The label SIDE-A is invalid because it contains the dash or minus sign. The third example is longer than six characters. The last example is not a proper label because it is identical to a language word.

A label is assigned by writing an equivalence statement. The general format of an equivalence statement is a label, followed by an equal sign, followed by a statement. Examples are:

```
N1 = 5
P1 = POINT/2, 3
```

The first statement assigns the label N1 to the value 5. When N1 is referred to later in a program statement, the value 5 is used. The second statement assigns the label P1 to the point value (2, 3), so that the programmer can reference the point as P1 in later statements without writing the coordinates.

New values can be assigned to labels in later program statements; in such cases, the new value completely replaces the old value. However, the new value must be of the same type as the original: if a label is used to indicate a point, the label can be assigned only to a new point value.

2.01.03 PUNCTUATION

Each word in a Drafting System language statement is separated from each other word by punctuation. Each character of punctuation has a specific meaning and must be used exactly as defined to produce correct statements. The characters of punctuation and their uses are:

Comma ,

The comma separates Drafting System language words, and user-created words that are not separated by other punctuation; the comma is the usual form of separation.

```
DOTTED, LINE/DX, 5
POINT/INTOF, LN1, LN2
CIRCLE/5, 3, 1
```

Double Comma ,,

This mark is used (1) to separate phrases in a CALL statement and (2) to separate statements written on the same line. The second feature enables the user to place more than one statement on a card.

```
CALL/RECT,,LEN=2,,WIDTH=4,,REF=POINT/2, 3
A=2,,B=3,,P2=POINT/5, 6
```

Equal Sign =

The equal sign is used to assign a value to a label. A statement in which an equal sign appears is called an equivalence statement.

```
V1 = VIEW/
N1 = 4-.74
P1 = POINT/N1, 5
L1 = LINE/P1,DX, 5
     END/V1
T2 = @MESSAGE@
```

Slash /

A slash follows each major word.

```
LINE/2, 3, 2, 5
POINT/4, 5
DRAW/FRAME
```

The slash is also used in arithmetic statements to indicate division.

```
A = 3/2
```

Plus Sign +

This sign is used to indicate the algebraic sign of a number. This use is optional since a number is assumed to be positive if no sign is present. The plus sign is also used to denote addition in arithmetic statements.

```
POINT/+2.3, +4.2
A = 6.4 + 10.9
```

Minus Sign -

The minus sign is used to indicate the algebraic sign of a scalar, or to denote subtraction in arithmetic statements.

```
POINT/-2.3, -4.2
A = 11.3 - 5.452
```

Asterisk *

This is used in arithmetic statements to indicate multiplication.

```
A = 2.6*4.3
```

Double Asterisk **

This is used in arithmetic statements to indicate that a quantity is to be taken to a given power. For example, the following statement sets B equal to the cube of two:

```
B = 2**3
```

Dollar Sign $

The dollar sign signifies that a statement is being continued on the next line or card; some other form of punctuation must precede the dollar sign.

```
LINE/P1, ATANGL, 30, $
TILLX, 3
```

The dollar sign is also used in literals that are created for notes. This feature enables a programmer to draw special drawing symbols by using a two-letter abbreviation enclosed in dollar signs.

```
NOTE/P1, 4, 7, @LENGTH 3 $FT$4 $IN$@
```

A detailed explanation of this feature of the language may be found in section 2.04.03.

Double Dollar Sign $$

This mark is used when the writer of a program wishes to place comments among the program statements. A comment can be a statement by itself:

```
$$BEGIN FRONT VIEW CODING
```

or can appear at the end of a regular language statement:

```
LINE/DX, 9 $$BOTTOM EDGE.
```

Parentheses ( )

Parentheses are used (1) to enclose nested definitions, function arguments, or the decision value in an IF statement, and (2) to indicate order and scope in an arithmetic statement.

```
POINT/INTOF, (LINE/DX, 8), L2
N5 = SQRT (7)
IF (N1 - 20) 1, 3, 3
N3 = (N1 + 5) * (N2 - 4)
```

Right Parenthesis)

This must follow each statement number. Numbers are given to certain statements to permit programmed looping. This feature of the language is explained in section 2.05.02.

```
1) LINE/DY, 3
```

Commercial At-Sign @

This symbol is used to enclose a user-created literal. The rules for creating literals are presented in section 2. 01. 02.

```
A = @ANY MESSAGE@
NOTE/P1, @CHAMFER 1/32 45 DEG@
```

Decimal Point .

The decimal point is used in writing numbers with a decimal fraction. When no decimal is indicated, a whole number is assumed.

```
2.36
 .047
```

Blank (the absence of a character)

Except in the case of literals, blanks are ignored by the Drafting System processor. The following two words are identical to the system:

```
LINE
L I N E
```

A blank is treated as a regular character within a literal. This enables the writer of a program to draw notes which have words separated by blank spaces.

```
NOTE/P3, 1, 1, @DO NOT GRIND EDGE@
```

2.01.04 STATEMENTS

The General Form

The words and punctuation of the language are combined to form statements. The general form of a statement is:

statement number) label = line class, major word/phrase

An example of a statement in this form is:

```
2) L1 = DASHED, LINE/DX, 7
```

The digit 2 is the statement number and is followed by a right parenthesis. The label L1 has been created by the programmer to represent the value of the line being defined in the statement. The language word DASHED indicates that a dashed line is desired. The major word LINE shows that a line is being defined. The phrase DX, 7 is one of the allowable forms for line definitions — the phrase defines a horizontal line seven inches long. The sample statement makes use of all the capabilities of the general form. However, most statements in a program use only portions of the general form.

Statement Numbers

As mentioned earlier, a statement number and its associated right parenthesis are written only in loops, and only when it is necessary to control the execution pattern of a set of statements. This feature of looping is explained in section 2.05.02.

Labels

The label and its accompanying equal sign are used only when it is desirable to be able to refer to a particular value more than once in a program. The following example shows how the point value (2, 6) is assigned to the label P4, so that the programmer can write P4, instead of the coordinates of the point, in later statements of the program,

```
P4 = POINT/2, 6
```

```
.
CIRCLE/P4, 3
.
.
DIM/P4, P5
```

Line Class

A geometric definition is given a line class by writing one of the language words reserved for this purpose. These words and their effects are discussed in section 2.02.01. Line class can also be used in lettering and cross-hatching.

The Major Word

A major word is necessary in all statements except arithmetic and literal equivalence statements. Examples of these two exceptions are:

```
N1 = 4.6 * N2
A8 = @SPACER, CLUTCH@
```

All other statements begin with a major word:

```
LINE/2, 2, 4, 6
DRAW/FRONT
TITLE/8, 2 @INFORMATION@
```

The Phrase

The words which can appear to the right of a major word depend upon the function of that major word. As each major word is discussed in the following sections of this manual, the words and phrases used with the major word will be defined.

Nesting

Normally, each statement in a program defines one item:

```
P2 = POINT/3, 6
     LINE/1, 5, P2
```

The first statement defines a point at coordinate (3, 6) and is given the label P2. The second statement defines a line between two points. The first point is at (1, 5), and the second point is the one labeled P2. The two statements can be combined into one by using a technique called nesting:

```
LINE/1, 5, (P2 = POINT/3, 6)
```

In this statement the definition of the point P2 is nested in the definition of the line. In general, parentheses are placed around a definition that is to be nested.

A statement may contain more than one definition. For example, the first point in the line definition can also be a nested definition:

```
LINE/(P3 = POINT/1, 5), (P2 = POINT/3, 6)
```

Furthermore, a nested definition can itself contain another nested definition. Suppose, for example, that P2 is the center of a circle and that it is desirable to label that circle as C5:

```
LINE/1, 5, (P2 = POINT/CENTER, (C5 = CIRCLE/3, 6, .5))
```

2.01.05 DOCUMENTATION CONVENTIONS

Throughout the remainder of this manual a standard presentation form is used in explaining language statements. There is a convention for introducing new statements and one for showing example statements.

1. Introduction of New Statements. A general form is shown each time a new statement is introduced. This is the general form for a line that is defined as the tangent to a circle from a given point outside of the circle:

$$\text{LINE/}\,\underline{\text{point}},\ \begin{matrix}\text{LEFT}\\ \text{RIGHT}\end{matrix},\ \text{TANTO},\ \underline{\text{circle}}$$

   Each language word is presented in capital letters. When a particular word can be one of several allowable language words, the options (LEFT or RIGHT in this example) are stacked.

   Besides language words, a statement also contains user-created words. In the general form, such variable words are printed in small letters and are underlined. The following list shows these words and what the programmer writes for each:

$\underline{\text{scalar}}$	A number or the label of a number, or an arithmetic expression that results in a number or the label of such an expression.
$\underline{\text{point}}$	The label of a point, a nested point definition, or two scalars separated by a comma.
$\underline{\text{point2}}$	The label of a point or any point definition within parentheses.
$\underline{\text{circle}}$	The label of a circle, a nested circle definition, a point and a scalar separated by a comma, or three scalars separated by commas.
$\underline{\text{circle2}}$	The label of a circle or any circle definition within parentheses.

line	The label of a line, a nested line definition, two points separated by a comma, or four scalars separated by commas.
line2	The label of a line or any line definition enclosed with parentheses.
arc	The label of an arc, a circle and two scalars separated by commas, or five scalars separated by commas.
view	Always the label of a view definition.
shape	Always the label of a shape definition.
alpha	A literal or the label of a literal.

2. Example Statements. After each new statement is introduced, example statements are presented. When appropriate, labels are used in the samples. The labels follow the standard pattern of a letter followed by a number. For points the first letter of the label is P, for circles it is C, for lines it is L, and for arcs it is A. The label of a scalar begins with N. The following examples illustrate this convention:

```
P1 = POINT/YLARGE, INTOF, L1, C3

A4 = ARC/C1, 45, N7
```

   This convention is intended only as an aid in learning the language. A programmer can create labels of any type as long as he follows the rules outlined in section 2.01.02.

2.02 GEOMETRIC STATEMENTS

The items on a drawing can be divided into two types: object lines and annotations. Object lines are those line segments, circles, and arcs which represent the object being drawn. Annotations consist of notes and dimensions. The concern here is with object lines — how they are defined and how they are grouped to form a picture. The following sample program illustrates the purpose of each section in this part of the manual:

```
1.  SQUAR = VIEW/
2.  LINE/DX, 1
3.  LINE/DY, 1
4.  LINE/DX, -1
5.  LINE/DY, -1
6.  DOTTED, CIRCLE/.5, .5, .25
7.  END/SQUAR
8.  DRAW/SQUAR
```

This program will draw a one-inch square in the middle of which is a circle having a quarter-inch radius. The numbers given to the statements are not part of the program statements, but are used only for explanation.

The object lines to be drawn are defined in statements 2 — 6. Statement 2, for example, defines a horizontal line one inch long. The language word DX indicates a horizontal line. There are several ways to define a line in the language. Section 2.02.03 presents each of the statements used to describe a line. The next two sections give the ways in which circles and arcs can be defined. Point definitions, explained in section 2.02.02, make it easier to describe the other geometric elements.

Statement 6 defines a circle with a center point of (.5, .5) and a radius of one-quarter inch. This statement illustrates the use of a line-class word. Defining the line class of geometric elements is an integral part of object-line programming and is discussed in section 2.02.01.

Statements 1 and 7 illustrate another feature of the language: the concept of grouping object lines. Statement 1, a VIEW statement, notifies the drafting processor that the object lines defined in following statements are to be grouped as a view and that the label SQUAR is to be assigned to that view. This grouping is terminated by statement 7, an END statement. The DRAW statement, number 8, indicates why items are grouped together. This statement causes all of the object lines within the view to be drawn. It is also possible to crosshatch the drawing by using the statement HATCH/SQUAR. Section 2.02.01 explains the details of grouping.

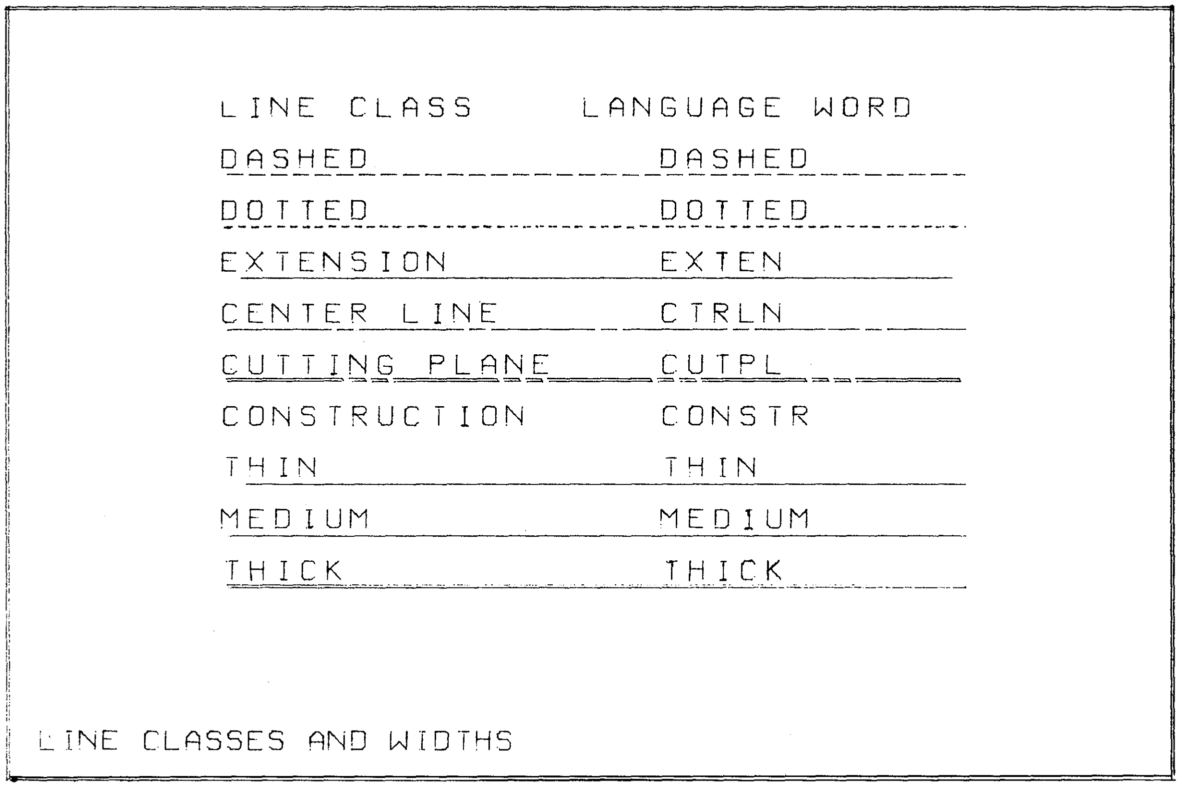
Section 2.02.06 describes geometric functions. These language features enable the programmer to do calculations with geometric elements. Included, for example, is a discussion of the way to calculate the distance between two points, using a single language word.

Section 2.02.07 explains the use of an auxiliary part reference system. This feature of the language makes it easier to code object lines when part of the object is at an angle to the main body of the view.

2.02.01 GROUPING

There are four aspects to the concept of grouping: (1) the setting of line classes, (2) the words used to indicate the beginning and end of a group, (3) the construction of a part model, and (4) the sequence of defining object lines.

1. Line Class. Figure 2 shows the various line classes and associated language words that are available in the 1620 Drafting System.



LINE CLASS	LANGUAGE WORD
DASHED	DASHED
DOTTED	DOTTED
EXTENSION	EXTEN
CENTER LINE	CTRLN
CUTTING PLANE	CUTPL
CONSTRUCTION	CONSTR
THIN	THIN
MEDIUM	MEDIUM
THICK	THICK

LINE CLASSES AND WIDTHS

Figure 2

There is no difference in the line weights produced by the words THIN and MEDIUM. Two words are made available so that a user can modify the 1620 Drafting System to draw different weights for these words if necessary. Details for making such a modification are contained in the section titled "Phase Three". Note also that a construction line, by definition, does not produce a drawn line.

The programmer can assign a line class to a line, circle, or arc when the element is defined:

```
DASHED, LINE/P1, P2
DOTTED, CIRCLE/P1, 4
MEDIUM, ARC/C1, 0, 180
```

A line class can also be assigned to an entire group of elements:

```
V1 = DOTTED, VIEW/
```

2. VIEW, SHAPE, and END statements. VIEW and SHAPE statements are used to indicate the beginning of geometric groups. The general form for a VIEW statement is:

   label = line class, VIEW/

   The general form for a SHAPE statement is:

   label = line class, SHAPE/

   The termination of a geometric group defined by a VIEW or SHAPE statement is indicated by an END statement. This statement has the following two forms:

   END/<u>view</u>

   END/<u>shape</u>

   Examples of these statements are:

```
FRONT = DASHED, VIEW/
FRONT = VIEW/
CAP = DOTTED, SHAPE/
      END/BACK
      END/CAP
```

   A VIEW or SHAPE statement is followed by statements which define object lines. This group of object line definitions is followed by an END statement. All of the object lines can then be referenced by writing the name given to the group, such as in the DRAW statement: DRAW/FRONT.

   VIEW and SHAPE statements differ in scope: a SHAPE grouping consists of individual geometric definitions, whereas a VIEW grouping may contain individual geometric elements and also items grouped by a SHAPE statement. This difference in scope allows flexibility in assigning line classes.

Four rules govern the line class of geometric definitions:

- A line class for an individual geometric definition affects only that statement.
- A line class specified for a SHAPE group affects all geometric definitions within that group except those that have their own line class specified.
- A line class specified in a VIEW statement affects all geometric definitions within that group except those within a SHAPE group or those individual elements that have a specified line class.
- When no line class word is written in a VIEW statement, the 1620 Drafting System assumes the MEDIUM line class. When the line class is omitted from a SHAPE statement, the 1620 Drafting System uses the line class of the previous VIEW statement.

The following example shows how these rules are applied:

Program Statement	Resultant Line Class	Reason
FRONT = DOTTED, VIEW/		
CT RLN, LINE/...	Center line	Rule 1
LINE/...	Dotted	Rule 3
SLOT = DASHED, SHAPE/		
LINE/...	Dashed	Rule 2
MEDIUM, CIRCLE/...	Medium	Rule 1
LINE/...	Dashed	Rule 2
END/SLOT		
LINE/...	Dotted	Rule 3
EXTEN, LINE/...	Extension	Rule 1
CIRCLE/...	Dotted	Rule 3
KEY = SHAPE/		
LINE/...	Dotted	Rules 2 & 4
ARC/...	Dotted	Rules 2 & 4
END/KEY		
END/FRONT		

3. The Part Model. Previous discussion has pointed out that no lines are drawn until a DRAW statement is given. The programmer first defines all of the object lines within a view or shape group and then writes a statement to draw the object lines. To do this, the 1620 Drafting System must save each object line as it is defined. The total set of object lines represents or models the part to be drawn. Details about the purpose or structure of the part model are unimportant in this discussion. However, a general understanding of how the part model affects drawing should be helpful to a programmer.

   First, no object line can be drawn unless it is in the part model. For the programmer, this means that object lines to be drawn must be within a view or a shape group. The following program illustrates this point:

```
L1  =  LINE/Any line definition
V1  =  VIEW/
L2  =  LINE/Any line definition
C1  =  CIRCLE/Any circle definition
END/V1
DRAW/V1
```

   In this example the line labeled L2 and the circle labeled C1 become part of the model and are drawn as a result of the DRAW command. The line labeled L1, however, is not defined as part of the model and, therefore, is not drawn.

   The example shows that it is necessary for an object line to be defined within a group in order to be drawn. This condition, however, is not sufficient — a line can be within a group (view or shape) and still not become a part of the model. This restriction applies to nested definitions. An example of the restriction appears in this program:

```
V1  =  VIEW/
L1  =  LINE/1, 2, 3, 4
L2  =  LINE/5, 6, (POINT/(L3 = LINE/3, 4, 4, 6))
END/V1
DRAW/V1
```

   In this case, lines L1 and L2 will be drawn. The nested line L3 will <u>not</u> be drawn. The rule which applies here is: Only the object line defined by the first major word in a statement can enter the part model. Nested definitions, therefore, never become part of the model. This rule applies also to the use of the double comma form of punctuation. The double comma separates statements written on one card:

```
L1  =  LINE/1, 2, 3, 4,,L2  =  LINE/5, 6, 7, 7,,L3  =  LINE/DX,4
```

   Only the line labeled L1 can enter the part model; L2 and L3 cannot be drawn.

There is one other point to consider. A line may be defined within a view or shape group; it may be defined by the first major word in the statement; and yet it may also be given a construction line class (the language word is CONSTR). The defined line will enter the part model but will not be drawn. Note that since a line definition can be nested or placed outside a view or shape group, and so not be drawn, a programmer has little use for the CONSTR line class. The primary reason for using the construction line class is in crosshatching. A crosshatch command (language word HATCH) does not draw object lines — it generates section lines across the object. In some instances it is necessary to draw only part of the object but to crosshatch all of it. In such cases the programmer assigns the CONSTR line class to the lines that are not to be drawn, and the DRAW command ignores those lines in the part model. The crosshatch command, however, uses the construction lines in determining crosshatch lines. Crosshatching is discussed fully in section 2. 04. 02.

4. Definition Sequence. Object lines are drawn in the same sequence in which they are defined.

   This fact alone usually is unimportant to a programmer. It is worthwhile, however, to consider the normal sequence in which a programmer defines object lines. The program statements to draw a square illustrate this point. Using the language of the 1620 Drafting System, it is necessary to define four lines in order to draw the square. A natural sequence for these line definitions would be to start at one corner, define a line from that corner, define the line that connects to the first line, and so on until the fourth line ended at the starting corner. The language has a feature that takes advantage of this natural sequence of description — the language word PPP.

   The word PPP represents the Present Part Position. As each object line is defined, the 1620 Drafting System keeps track of the ending point of that object line. For example, when a line is defined in a statement, the system saves the ending point of the line. This ending point is available to the programmer through the language word PPP. This word is a label of a point — the present part position. Anywhere a point can occur in a language statement, a programmer can use the label PPP. Through this word it is possible to take advantage of the natural tendency to define lines that connect to one another.

   Each geometric definition changes the present part position. When a point is defined, that point becomes PPP. When a line is defined, the ending point of the line becomes PPP. When a circle is defined, the present part position becomes the point on the circumference directly to the right of the center point. When an arc is defined, the ending point of the arc becomes PPP. A VIEW statement also affects PPP: When a view group begins, the present part position is set at (0, 0). A SHAPE statement does not affect PPP.

2. 02. 02 POINT DEFINITIONS

The definition of a point cannot cause drawing. Points are defined to aid in defining items that can be drawn — lines, circles, and arcs. A line, for example, can be defined by its two end points. The programmer may wish to predefine the two points and use the labels of the points in writing the line statement:

```
P1 = POINT/ 4, 6

P2 = POINT/ 9, 11.4

     LINE/ P1, P2
```

Using point definitions in this manner, the programmer always assigns a label to a point definition. In another use for point statements — in connection with the operation of the Present Part Position, PPP — no label need be given. Whenever the word POINT is the first major word in a statement, the value of that point definition becomes the value of PPP. The following two statements illustrate such a use:

```
POINT/INTOF, L1, L2

LINE/PPP, 3, 7
```

The general form for all point statements is:

statement number) label = POINT/phrase

As each point definition is explained in the following paragraphs, the general form of the phrase will be given.

POINT — Rectangular Coordinates

General Form:

POINT/ <u>scalar</u>, <u>scalar</u>

A point may be defined by its coordinates. The two scalars are the X and Y values in a two-dimensional rectangular coordinate system. This is the standard (or canonical) form of a point definition — the Drafting System processor converts all point definitions to this form when manipulating point values.

Figure 3 depicts the following statements:

```
P1 = POINT/2, 3

P2 = POINT/2 + 1/2, 1

P3 = POINT/5,2

P4 = POINT/3, -1

P5 = POINT/ -1, -.5

P6 = POINT/ -1, 1.5
```

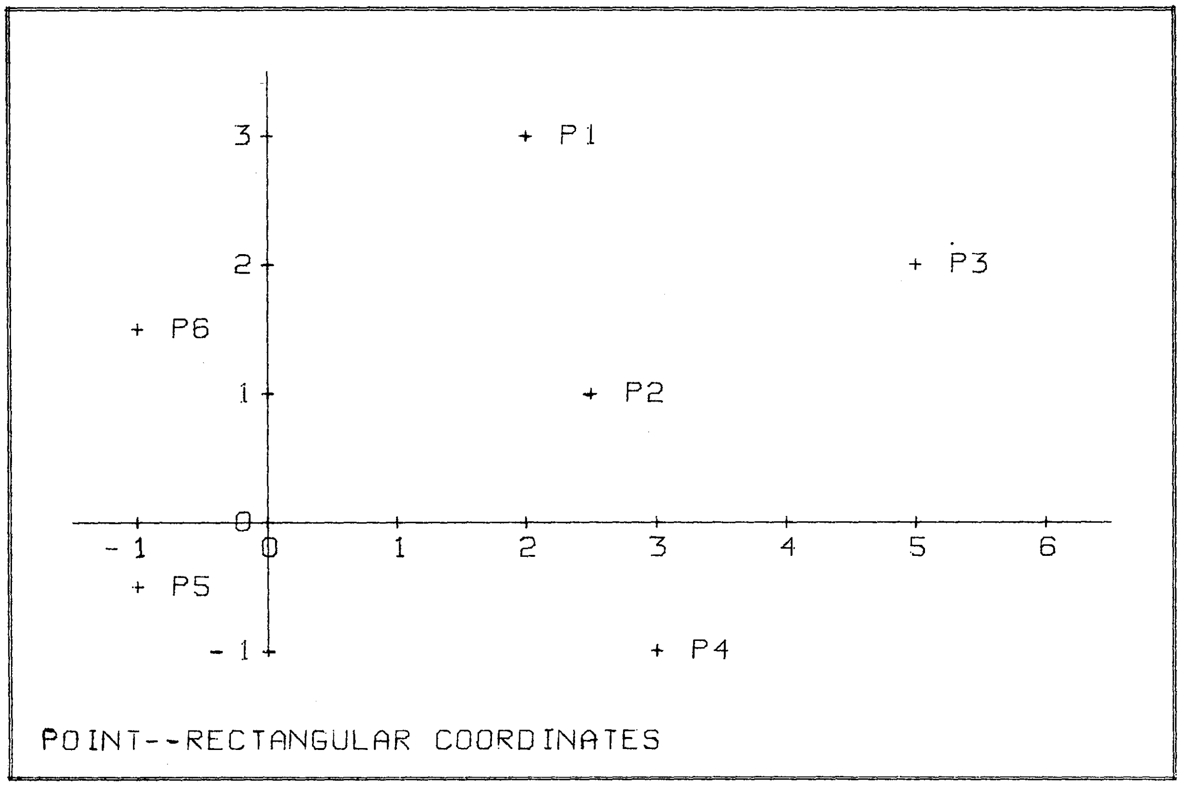


Figure 3

POINT — Intersection of Two Lines

General Form:

POINT/INTOF, line, line 2

The language word INTOF indicates "intersection of".

A point may be defined by the intersection of two nonparallel lines. All line definitions in the 1620 Drafting System are line segments — they have a beginning point and an ending point. However, in processing this statement, the 1620 Drafting System treats the line segments as being infinitely long and computes the intersection point of the two extended line segments. It is an error to specify parallel lines; therefore, should they be given, an error message is typed out during processing.

In Figure 4, L1 and L2 are line segments. The point labeled P1 is the intersection of the two lines. The dashed lines show how the 1620 Drafting System treats the lines as being infinitely long. The intersection of lines L3 and L4 defines the point P2.

Coding examples for Figure 4:

```
P1 = POINT/INTOF, L1, L2

P2 = POINT/INTOF, L3, L4
```

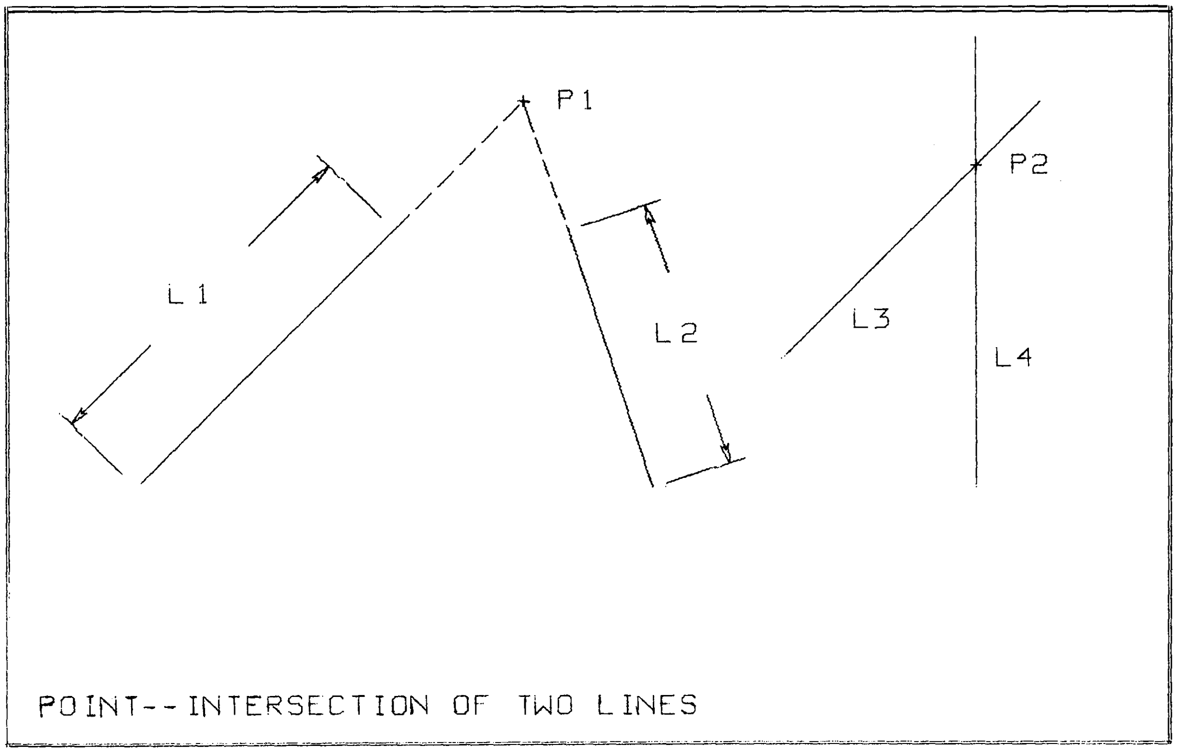


Figure 4

POINT — From other Geometric Definitions

General Forms:

POINT/point

POINT/line

POINT/circle

POINT/arc

POINT/point is the basic equivalence statement for a point:

P1 = POINT/PPP

P3 = POINT/P2

POINT/line allows the programmer to use any line definition to define a point. The point defined is the second point of the line. For example, if a line is defined:

L1 = LINE/3, 5, 7, 9

the statement: P1 = POINT/L1

defines P1 as the point (7, 9).

A line definition can be within the point statement:

P5 = POINT/LEFT, TANTO, C1, RIGHT, TANTO, C2

POINT/circle extracts the center point from a circle. The optional form POINT/CENTER, circle may also be used.

P4 = POINT/C1

P2 = POINT/CENTER, C1

P6 = POINT/CIRCLE/P1, P2, P3

POINT/arc is used to obtain the center point of an arc. The optional form POINT/CENTER, arc may also be used.

P8 = POINT/A3

P9 = POINT/ARC/P1, P5, XLARGE, RADIUS, 1

2.02.03 LINE DEFINITIONS

All line statements in the 1620 Drafting System define bounded line segments — the lines have a beginning point and an ending point. Lines are defined as segments so that they can be drawn. However, when geometric constructions are performed — such as the intersection of two lines — the lines are treated by the processor as being infinitely long.

A line definition affects the present part position, PPP. When LINE is the first major word in a statement, PPP is set equal to the ending point of the line.

All line statements have the following general form:

statement number) label = line class, LINE/phrase

The content of the phrase for each allowable line definition is specified in the paragraphs that follow.

LINE — From a Point to a Point

General Form:

LINE/point, point

A line may be defined from a point to a point. This represents a directed line segment; when the line is drawn, it is drawn from the first point to the second point. The two-point form is the standard line definition; all line definitions in the 1620 Drafting System are converted to the two-point form by the processor.

Coding examples for Figure 5:

```
P1 = POINT/3, 3
P2 = POINT/6, .5
L1 = LINE/ -1, .5, P2
L2 = LINE/P2, 6, 2
L3 = LINE/PPP, P1
L4 = LINE/P1, -1, 1
L5 = LINE/ -1, 1, -1, .5
DASHED, LINE/P1, P2
```

LINE — Relative Distance

General Forms:

LINE/point, DX, scalar

LINE/point, DY, scalar

LINE/point, DX, scalar, DY, scalar

DX is a language word indicating that the number following it is a distance measured along the X-axis.

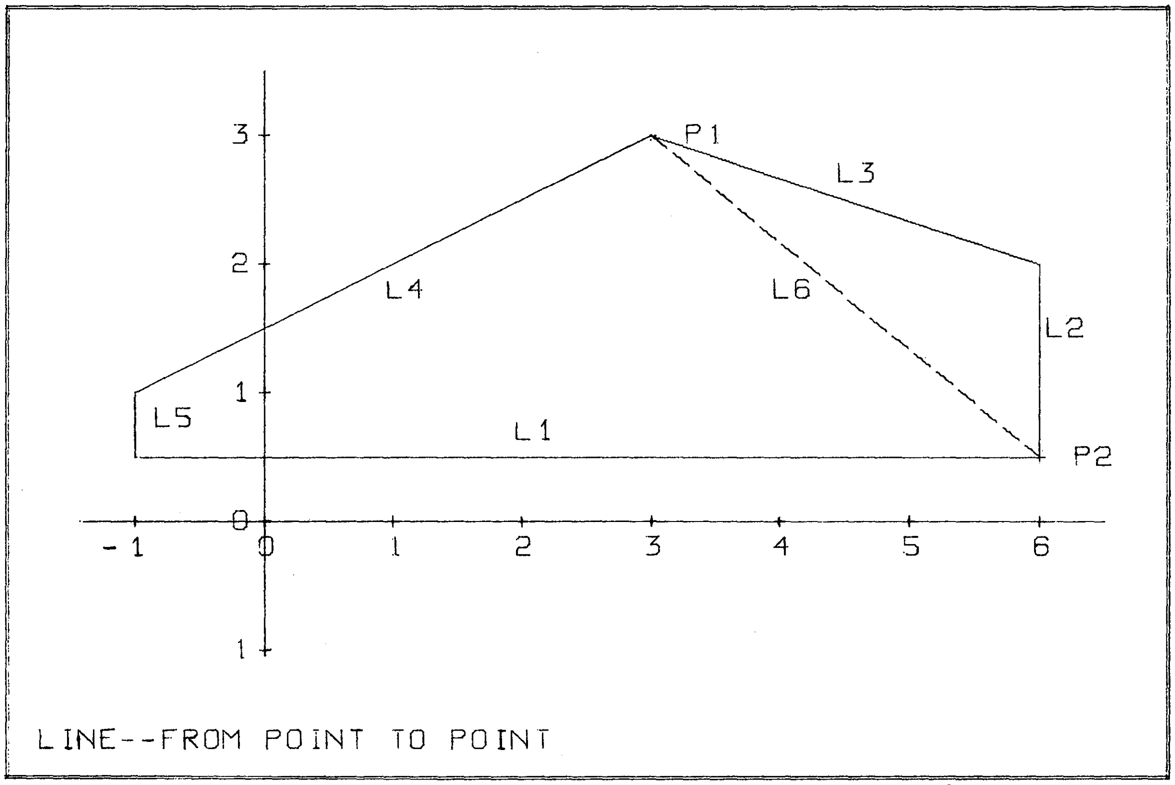


Figure 5

DY is a language word indicating that the number following it is a distance measured along the Y-axis.

The relative-distance form of line definition specifies a line by its end points. The first point is given in the statement. The second point is specified by its distance from the first point. The distance is measured by its horizontal component, vertical component, or both.

Line statements may be written to define horizontal, vertical, or oblique lines. Figure 5 illustrates the following examples:

```
L1 = LINE/ -1, .5, DX, 7
L2 = LINE/P2, DY, 1.5
L3 = LINE/PPP, DX, -3, DY, 1
L4 = LINE/P1, DX, -4, DY, -2
L5 = LINE/PPP, DY, -.5
DASHED, LINE/P1, DX, 3, DY, -2.5
```

<u>LINE — Distance from Present Part Position</u>

General Forms:

LINE/DX, <u>scalar</u>

LINE/DY, <u>scalar</u>

LINE/DX, <u>scalar</u>, DY, <u>scalar</u>

DX is a language word indicating that the number following it is a distance measured along the X-axis.

DY is a language word indicating that the number following it is a distance measured along the Y-axis.

This form of line definition specifies a line whose beginning point is the present part position, PPP. The form is similar to the relative-distance line definition; for example, LINE/DX, <u>scalar</u> is equivalent to LINE/PPP, DX, <u>scalar</u>. The form described here is provided in the language because many line definitions begin at the ending point of the previous line.

Coding examples for Figure 5:

```
     POINT/ -1, .5
L1 = LINE/DX, 7
L2 = LINE/DY, 1.5
L3 = LINE/DX, -3, DY, 1
L4 = LINE/DX, -4, DY, -2
L5 = LINE/DY, -.5
     POINT/P1
DASHED, LINE/DX, 3, DY, -1
```

LINE — From a Point at an Angle to the X-axis

General Forms:

```
                                  LENGTH
LINE/point, ATANGL, scalar, TILLX, scalar
                                  TILLY
                      LENGTH
LINE/ANGOF (line), TILLX, scalar
                      TILLY
```

A line may be defined from a point at an angle to the positive X-axis and bounded by a length, an X-coordinate, or a Y-coordinate.

ATANGL	Indicates that the following scalar is the angle in degrees the line makes with the positive X-axis. Counterclockwise is considered plus; clockwise is considered minus.
LENGTH	Indicates that the following scalar is the length of the line from the point specified.
TILLX	Indicates that the following scalar is an X-coordinate, and that the line is terminated at that coordinate. It is an error to specify a line parallel to the Y-axis, but if it is specified, an error message is typed during processing.
TILLY	Indicates that the following scalar is a Y-coordinate. It is an error to specify a line parallel to the X-axis, but if it is specified, an error message is typed during processing.

ANGOF (line) may be substituted for point, ATANGL, scalar. The first point of the defined line is the same as the first point of the given line. The angle of the defined line is the same as the angle of the given line.

Coding examples for Figure 6:

```
L1 = LINE/P1, ATANGL, 30, TILLX, 3.5
P3 = POINT/P1, ATANGL, 30, TILLX, 3.5
L2 = DOTTED, LINE/P1, ATANGL, 30, TILLX, 0
L3 = LINE/4, 2, ATANGL, 45, LENGTH, 2
L4 = DASHED, LINE/ANGOF (L3), LENGTH, -2
L5 = LINE/P2, ATANGL, 30, TILLY, 1.5
```

LINE — Intersection of a Line and a Circle

General Form:

```
       XLARGE
LINE/XSMALL, INTOF, line, circle 2
       YLARGE
       YSMALL
```

A line can be defined as being bound by a circle. Two points are produced when a line intersects a circle; the modifying word before INTOF selects which of the two points is to be the second point of the line. The modifier selects the point based on the X- or Y-values of the points. When the modifier is omitted, the processor selects the XLARGE point.

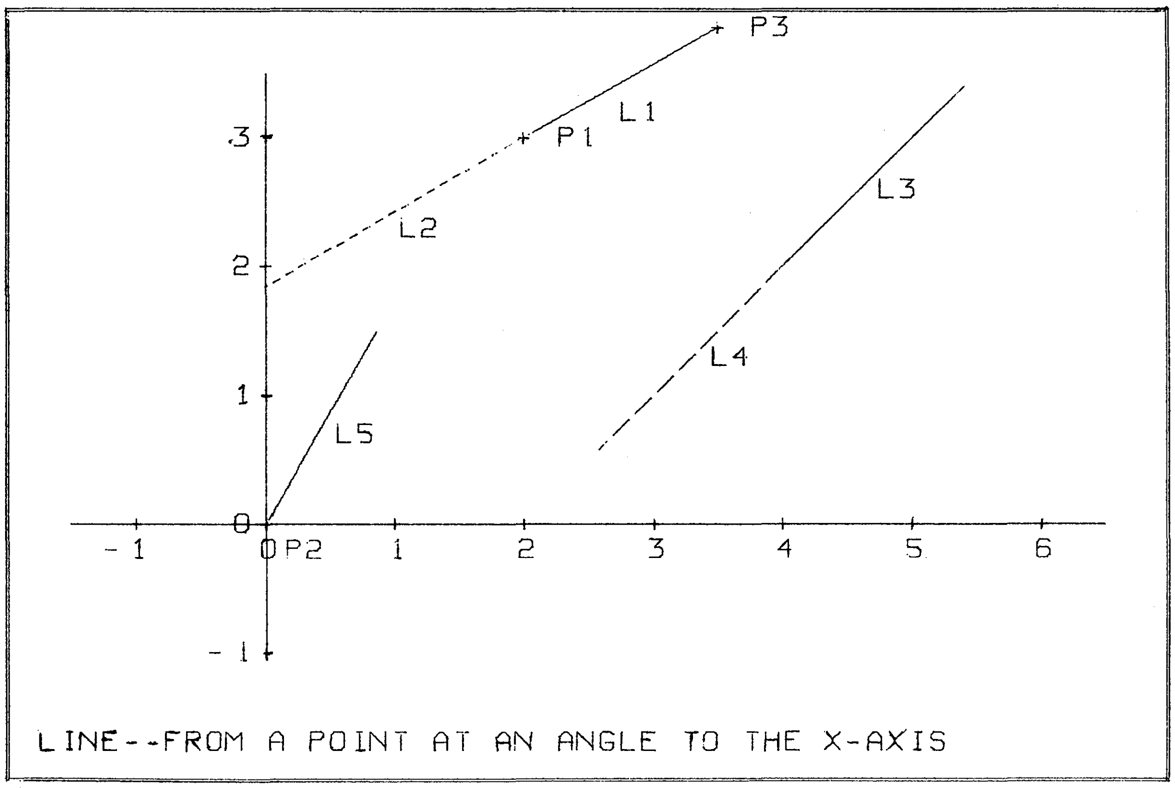


Figure 6

XLARGE	Indicates that the point with the largest X-value is to be selected.
XSMALL	Indicates that the point with the smallest X-value is to be selected.
YLARGE	Indicates that the point with largest Y-value is to be selected.
YSMALL	Indicates that the point with the smallest Y-value is to be selected.
INTOF	Indicates "intersection of".

An error message is printed automatically by the processor if the line does not intersect the circle.

Coding examples for Figure 7:

```
L2 = LINE/XSMALL, INTOF, L1, C1
L3 = LINE/XLARGE, INTOF, L1, C2
P2 = POINT/XSMALL, INTOF, L1, C2
P1 = POINT/INTOF, L1,C1
```

<u>LINE — Intersection of Two Circles</u>

General Form:

```
       XLARGE
  LINE/XSMALL, INTOF, circle, circle 2
       YLARGE
       YSMALL
```

A line can be defined by the intersection of two circles. Two points are produced when two circles intersect; the modifying word before INTOF selects which of the two points is

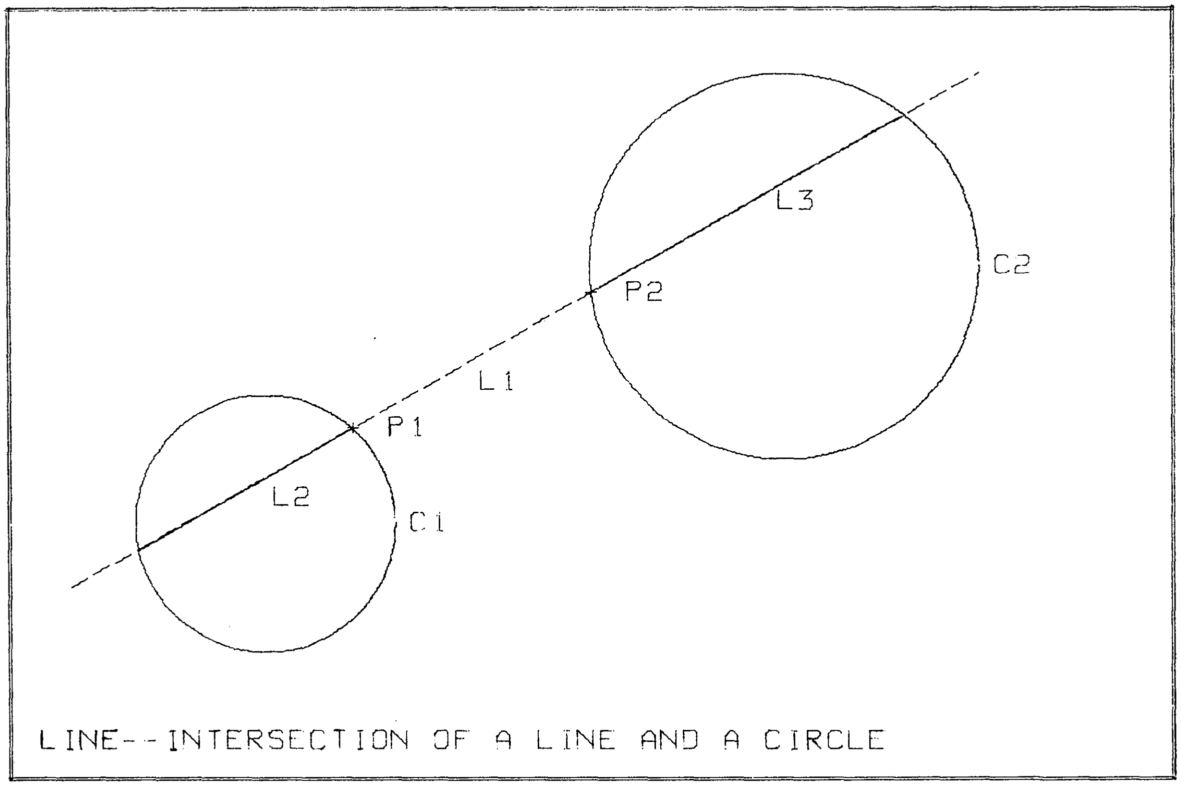


Figure 7

to be the second point of the line. The modifier selects the point based on the X- or Y-values of the points. When the modifier is omitted the processor selects the XLARGE point.

XLARGE	Indicates that the point with the largest X-value is to be selected.
XSMALL	Indicates that the point with the smallest X-value is to be selected.
YLARGE	Indicates that the point with the largest Y-value is to be selected.
YSMALL	Indicates that the point with the smallest Y-value is to be selected.
INTOF	Indicates "intersection of".

An error message is printed automatically by the processor if the two circles do not intersect.

Coding examples for Figure 8:

```
L1 = LINE/YSMALL, INTOF, C1, C2
P1 = POINT/YSMALL, INTOF, C1, C2
P2 = POINT/YLARGE,INTOF, C1, C2
```

<u>LINE — From a Point Tangent to a Circle</u>

General Form:

```
         LEFT
LINE/point, RIGHT, TANTO, circle
```

A line segment can be defined tangent to a circle from a point outside the circle.

TANTO	Indicates a tangent is to be solved.

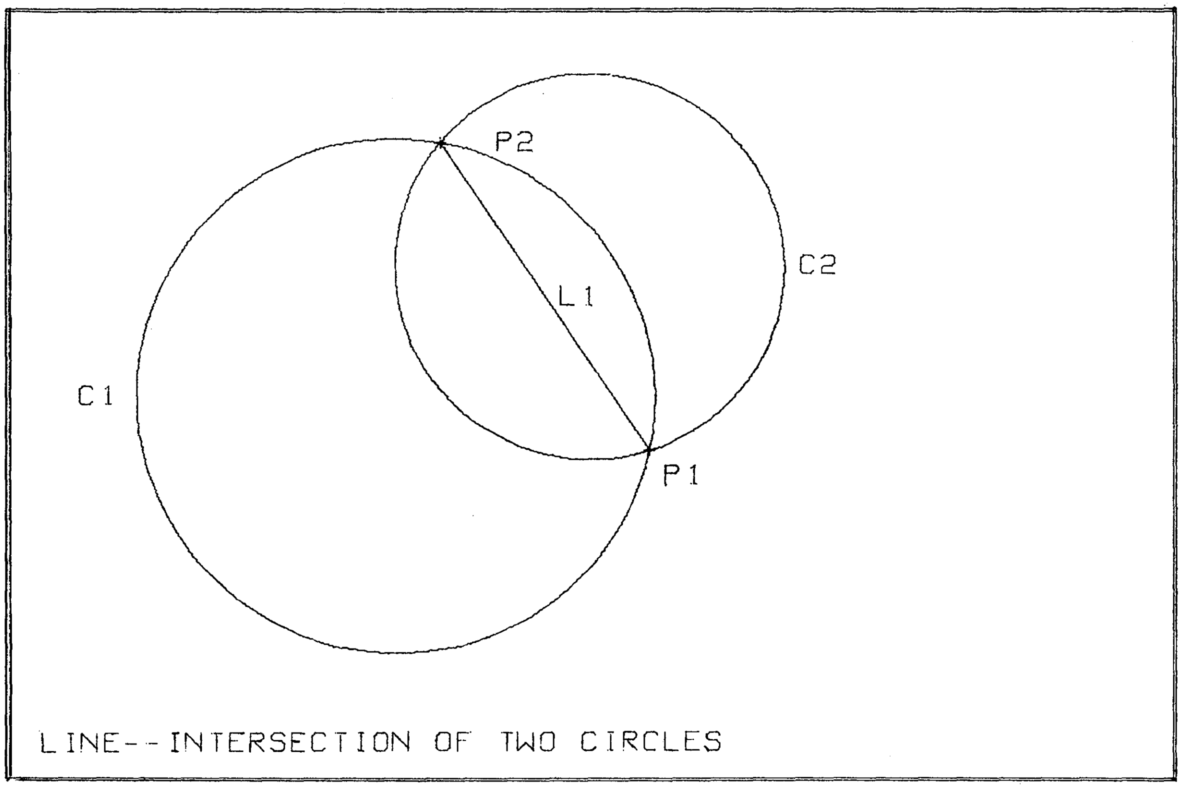


Figure 8

LEFT RIGHT	Indicates which of the two possible tangencies should be solved; the convention of selection is looking at the center of the circle from the given point. If no modifier is written, the processor selects the LEFT condition.

When the given point lies inside the circle or on the circumference, the processor prints an error message.

Coding examples for Figure 9:

```
L1 = LINE/P1, LEFT, TANTO, C1
L2 = LINE/P1, RIGHT, TANTO, C1
P3 = POINT/P1, TANTO, C1
P4 = POINT/P1, RIGHT, TANTO, C1
```

<u>LINE — Tangent to Two Circles</u>

General Form:

```
       LEFT                       LEFT
LINE / RIGHT, TANTO, circle, RIGHT, TANTO, circle
```

A line can be defined tangent to two given circles. This definition produces a line segment from the first tangency to the second. The modifier to TANTO should be written as if the observer were positioned at the center point of the first circle looking toward the center of the second circle. When the modifier is omitted, the LEFT modifier is selected by the processor.

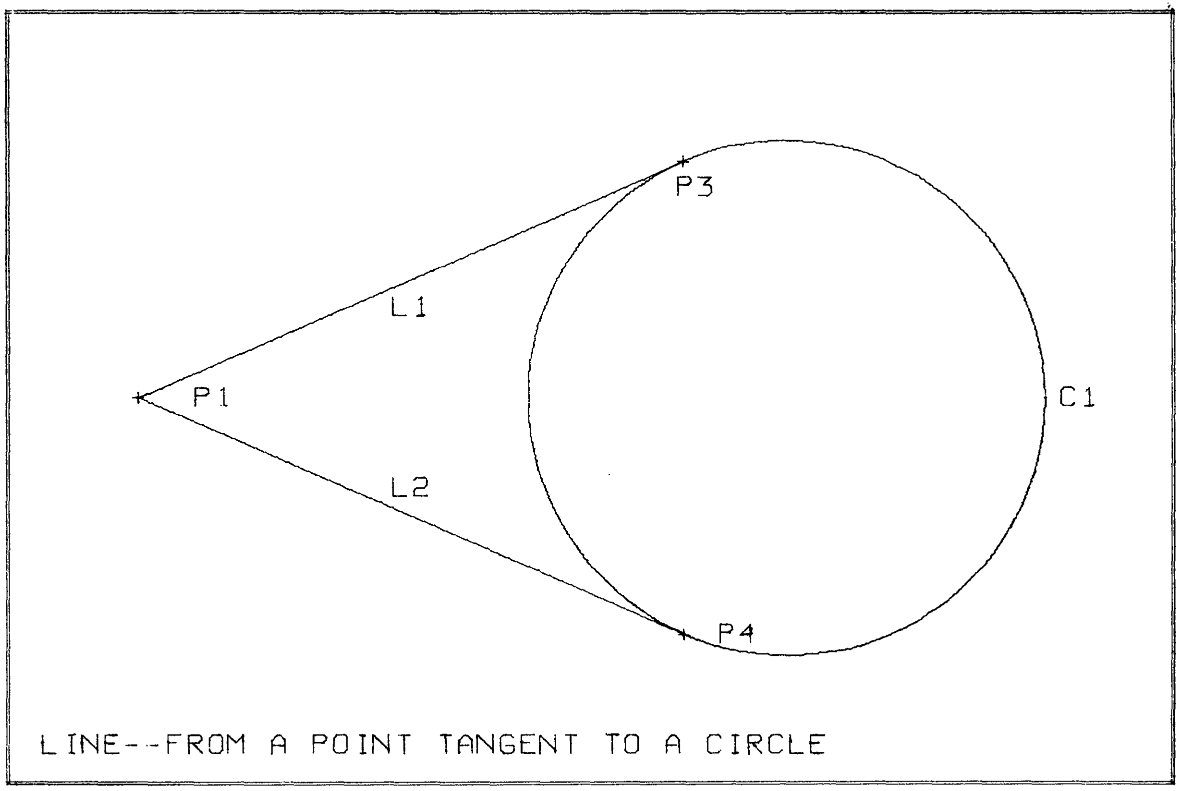


Figure 9

When the two circles intersect, the modifier for both tangencies must be either LEFT or RIGHT. Left-to-right or right-to-left combinations do not exist, and if used, cause the Drafting System processor to type out an error message.

When one circle lies within the other circle entirely, no solution exists, and the processor types out an error message.

Coding examples for Figure 10:

```
L1 = LINE / RIGHT, TANTO, C1, LEFT, TANTO, C2
L2 ≐ LINE / RIGHT, TANTO, C2, RIGHT, TANTO, C1
P1 = POINT / RIGHT, TANTO, C1, TANTO, C2
P2 = POINT / RIGHT, TANTO, C2, RIGHT, TANTO, C1
```

LINE — Parallel to a Line through a Point

General Form:

LINE/point, PARLEL, line

The language word PARLEL indicates parallel.

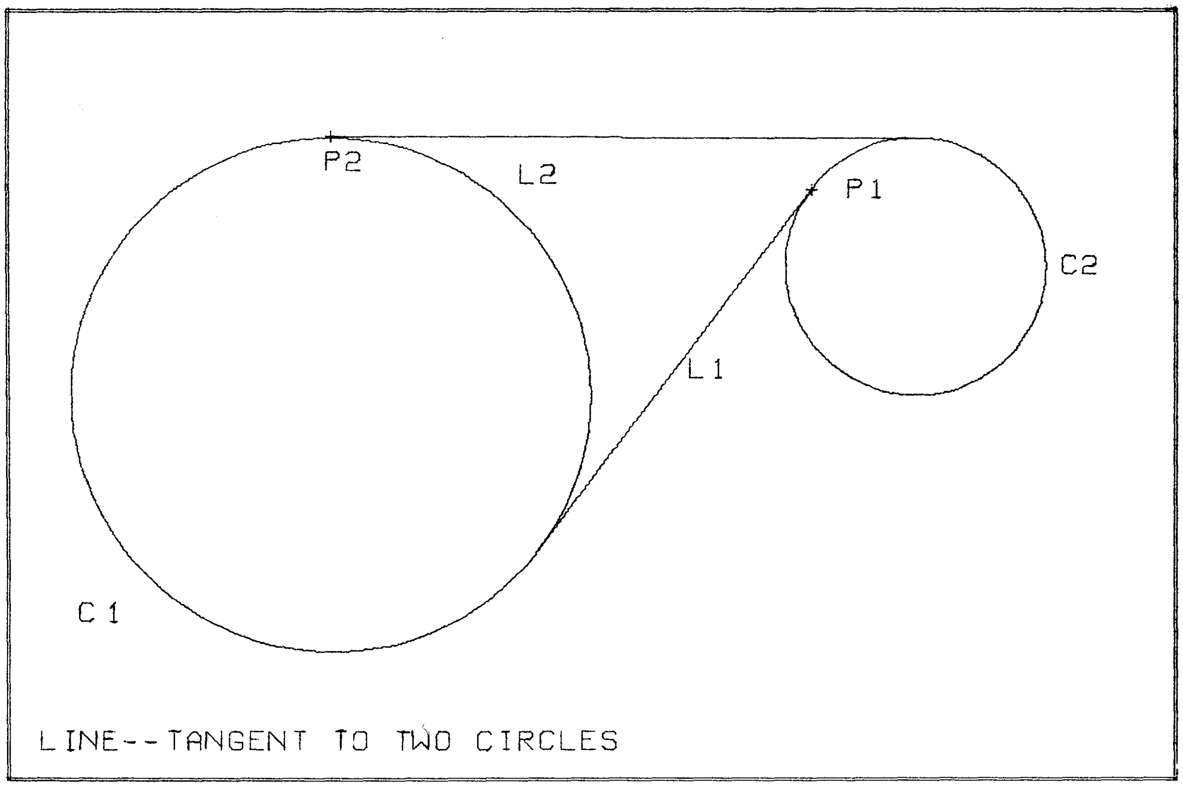


Figure 10

This definition produces a line parallel to a given line through a given point. The beginning point of the defined line is the given point. The ending point of the defined line is calculated in one of two possible ways.

1. If the given line is vertical, the second point of the parallel line has a Y-value ten inches greater than the Y-value of the given point.

2. If the given line is not vertical, the second point of the parallel line has an X-value ten inches greater than the X-value of the given point. Because of this arbitrary way of selecting the second point of the line, a programmer normally uses this line definition for construction purposes only. However, if appropriate, the line may be drawn.

Figure 11 shows the following examples:

```
L3 = LINE/ P2, PARLEL, L2

L4 = LINE/ P1, PARLEL, L1
```

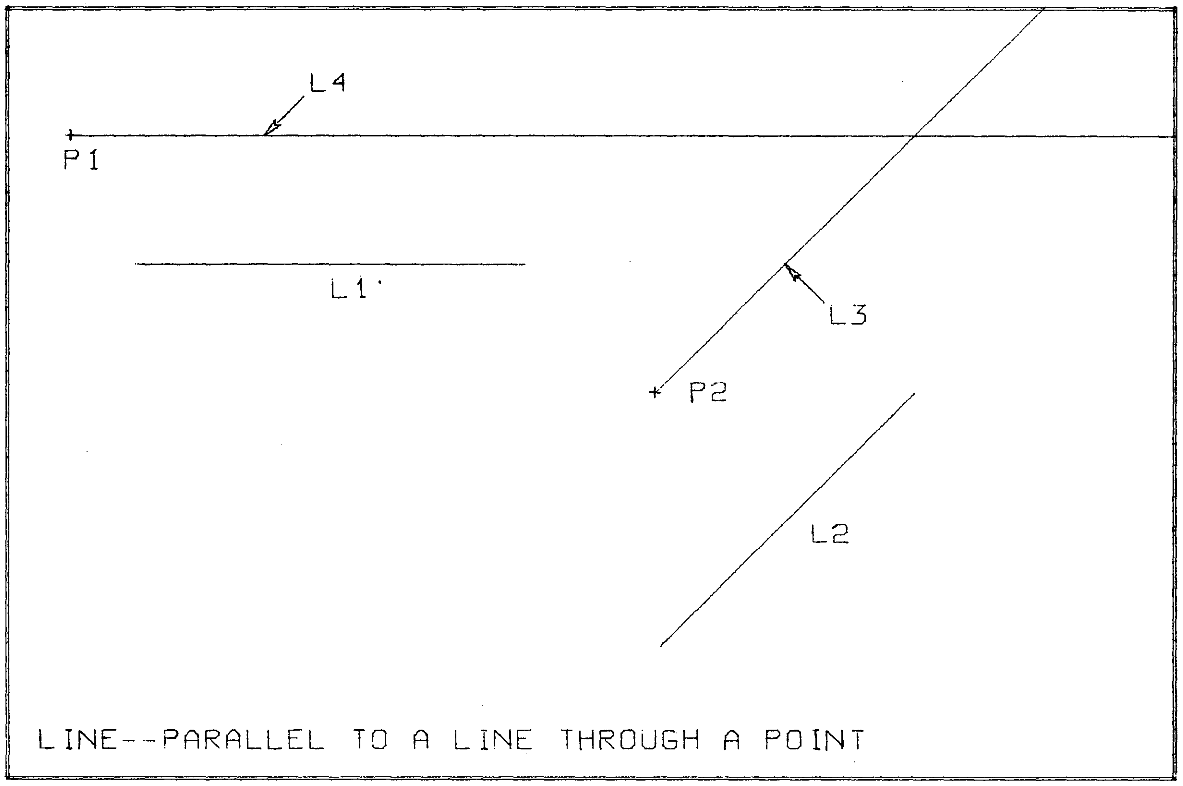


Figure 11

LINE — Parallel to a Line Offset a Distance

General Form:

```
       XLARGE
LINE/  XSMALL,  PARLEL,  line,  scalar
       YLARGE
       YSMALL
```

The language word PARLEL indicates parallel.

A line can be defined parallel to a given line and offset a given distance. The distance (defined by the scalar) is measured along a line perpendicular to the two lines. One of the four modifying words, before PARLEL, selects which of the two possible parallel lines is desired. The modifier may be chosen in either of two ways. (1) If, for a given X-value, the Y-value of the defined line is greater than the Y-value of the given line, the proper word is YLARGE. (2) If, for a given Y-value, the X-value of the defined line is greater than the X-value of the given line, the proper word is XLARGE. If the modifier is omitted, the processor uses XLARGE.

The end points of the defined line are calculated in one of two ways. (1) If the given line is vertical, the end points of the defined line have the same Y-values as the given line. (2) If the given line is not vertical, the end points of the defined line have the same X-values as the end points of the given line.

The following examples, in Figure 12, show the results of this definition:

```
L2 = DASHED, LINE/YLARGE, PARLEL, L1, .75

L3 = LINE/YSMALL, PARLEL, L1, 1

L4 = LINE/XSMALL, PARLEL, L5, .5

L7 = LINE/PARLEL, L6, 1.25
```

LINE — Perpendicular to a Line from a Point

General Form:

LINE/ point, PERPTO, line

The language word PERPTO indicates "perpendicular to".

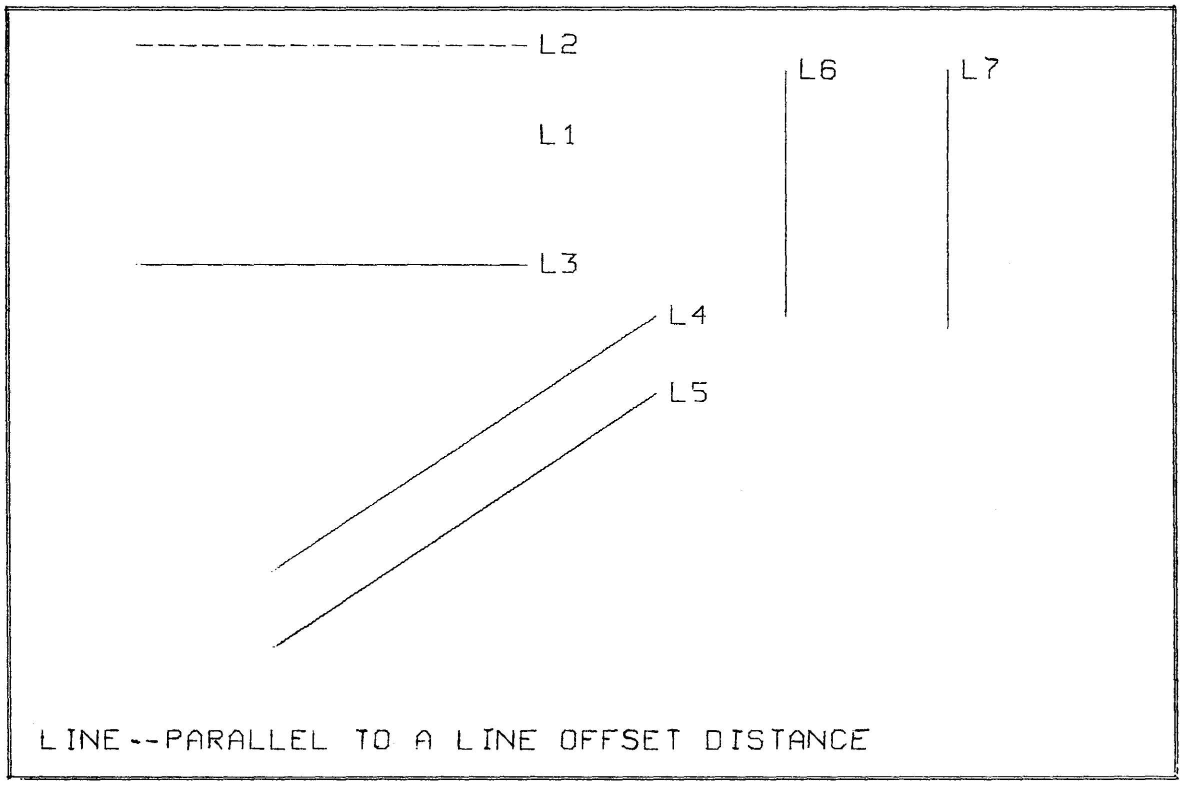


Figure 12

A line may be defined perpendicular to a given line from a given point. The beginning point of the resultant line is the given point; the ending point is the intersection of the two perpendicular lines.

If the given point lies on the given line, no solution is possible; the processor, therefore, types out an error message.

The following definitions are illustrated in Figure 13:

L3 = DASHED, LINE/P1, PERPTO, L1

L4 = LINE/ P2, PERPTO, L1

2.02.04 CIRCLE DEFINITIONS

These statements can be used for drawing circles. Circles may also be defined for the construction of other lines on a drawing. The general form for all circle statements is:

statement number) label = line class, CIRCLE/phrase

The allowable phrase for each circle definition is explained in the parts of this section that follow.

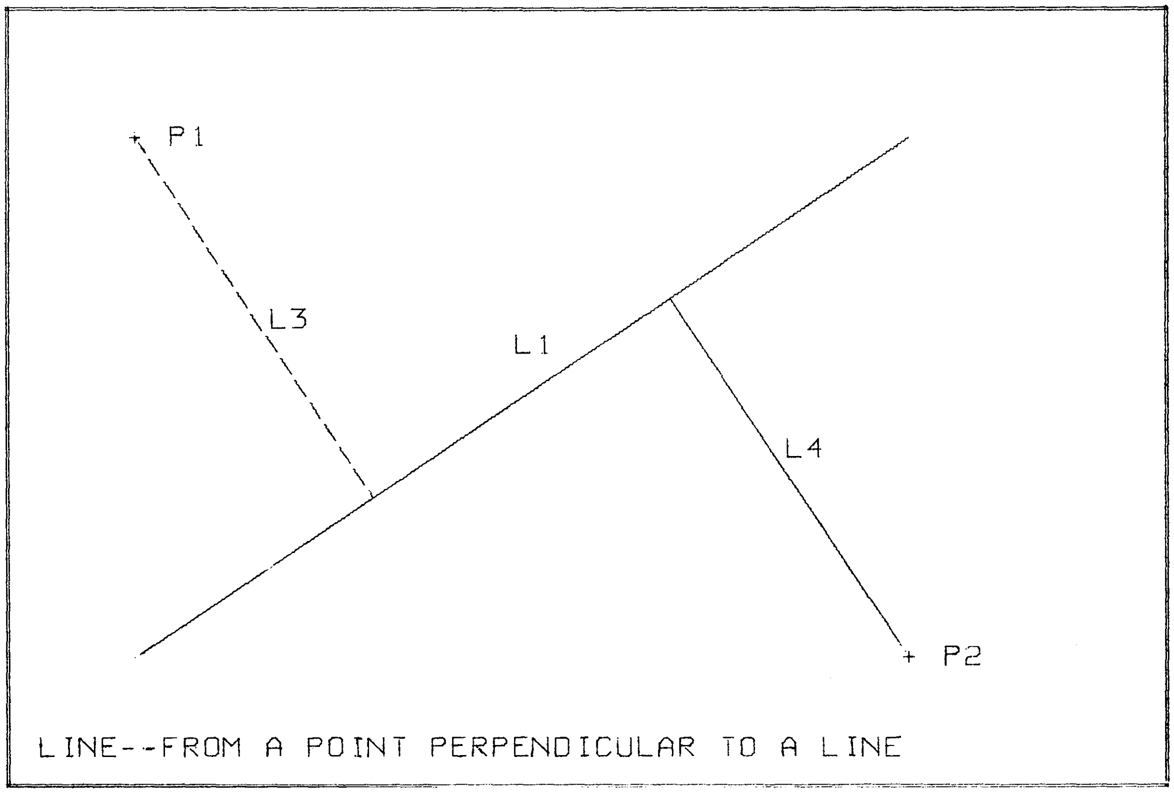


Figure 13

The Present Part Position, PPP, is updated by a circle definition. When CIRCLE is the first major word in a statement, PPP is set equal to the point on the circle to the right of the center point — that is, (XC + R, YC).

CIRCLE — Center Point and Radius

General Form:

CIRCLE/ point, scalar

point is the center of the circle

scalar is the radius of the circle

This is the basic form of circle definition. The 1620 Drafting System, when processing circles, converts all circles to this form.

The following statements are depicted in Figure 14:

```
C1 = DOTTED, CIRCLE/2, 2, 1.5

C2 = CIRCLE/ 2, 2, .8

C3 = CIRCLE/ -1, 1.5, .5

C4 = DOTTED, CIRCLE/ 5.5, 1.5, 1
```

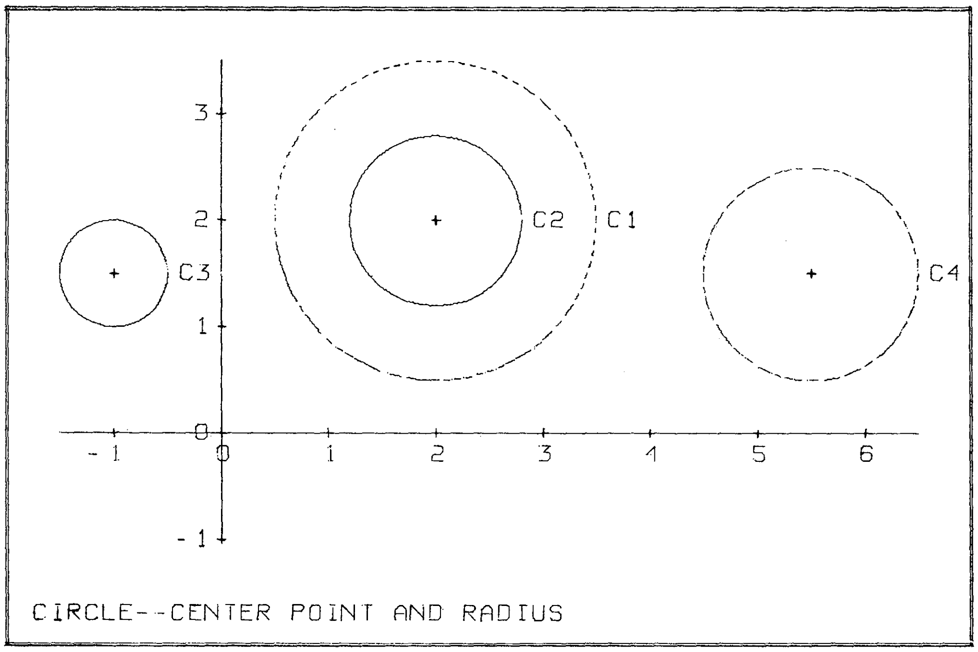


Figure 14

CIRCLE — Center and Radius of an Arc

General Form:

CIRCLE/ arc

An arc can be converted into a circle by using this statement form. The defined circle has the same center point and radius as the given arc.

Example statements: CIRCLE/ A4

DIMCR/ (CIRCLE/A3), 45

CIRCLE — Defined by Three Points

General Form:

CIRCLE/point, point, point 2

A circle may be defined by specifying three points through which the circle passes. When the three given points lie on a straight line, there is no solution; the processor, therefore, types out an error message. Figure 15 illustrates the statement:

C1 = CIRCLE/P1, P2, P3

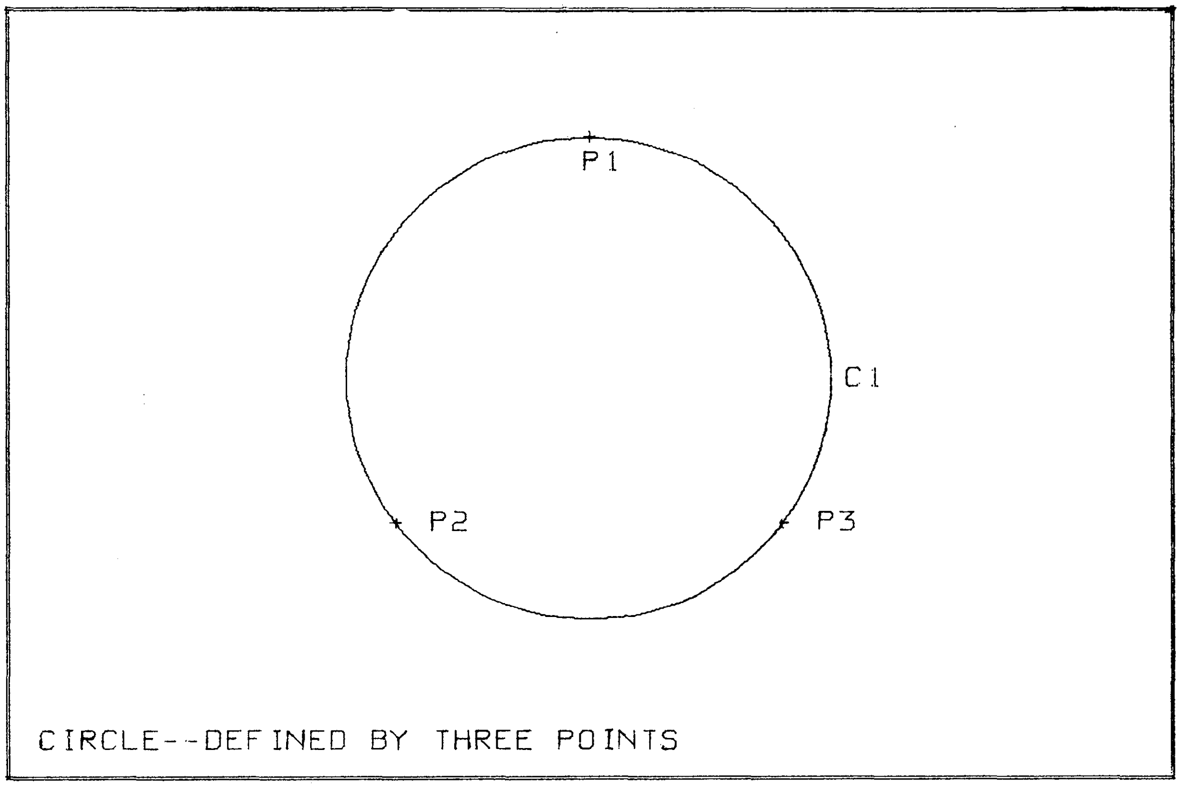


Figure 15

CIRCLE — Tangent to Two Lines

General Form:

```
              XLARGE                     XLARGE
       CIRCLE/XSMALL,  TANTO,  line,  XSMALL,  TANTO,  line,  scalar
              YLARGE                     YLARGE
              YSMALL                     YSMALL
```

TANTO is a language word that indicates "tangent to".

scalar is the radius of the desired circle.

A circle with a known radius can be defined tangent to two nonparallel lines. The four language words before TANTO are used to select the side of the line on which the center of the circle is to lie. When a modifier is omitted, the processor assumes XLARGE. The modifier may be chosen by comparing the center of the circle with the point of tangency. For example, if the Y-coordinate of the center point is higher than the Y-coordinate of the point of tangency, the modifier is YLARGE.

The processor produces an error message if the specified lines are parallel.

The following two circles are shown in Figure 16:

C1 = CIRCLE/YLARGE, TANTO, L1, YSMALL, TANTO, L2, 1.0

C2 = CIRCLE/TANTO, L3, YSMALL, TANTO, L4, 1.5

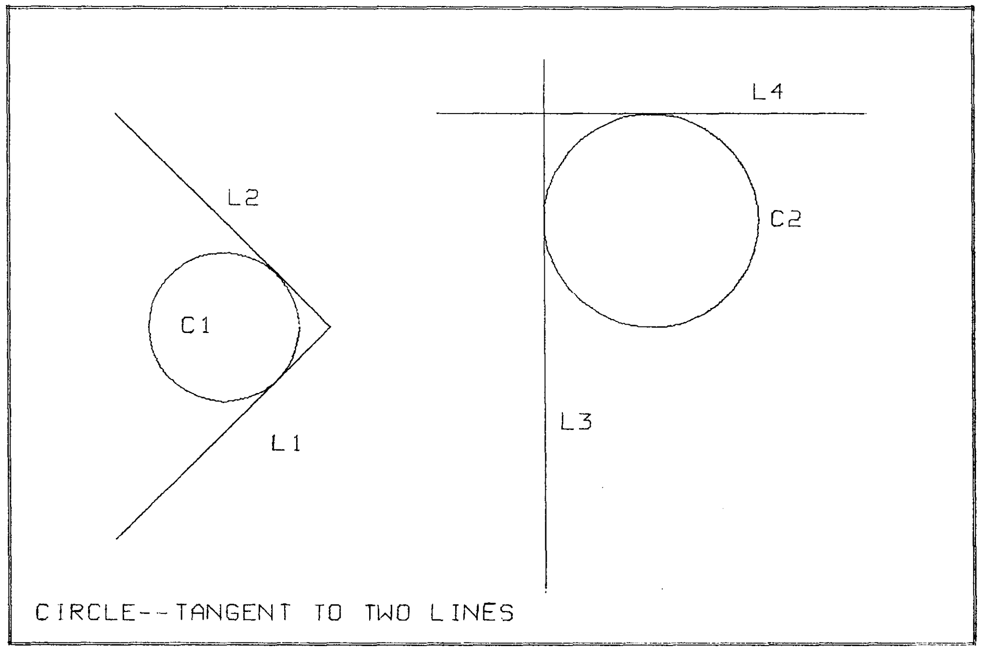


Figure 16

2.02.05 ARC DEFINITIONS

The general form for statements which define circular arcs is:

statement number) label = line class, ARC/phrase

The phrase used in each arc definition is defined in the parts of this section which follow.

The present part position, PPP, is updated by an ARC statement. When the word ARC is the first major word in a statement, PPP is set to the ending point of the arc.

ARC — Basic Definition

General Form:

ARC/ point, scalar, scalar, scalar

This is the standard arc definition, and all other definitions are converted to this form by the Drafting System processor.

The programmer specifies the following items in the sequence given: the center point of the arc; the radius of the arc; the beginning angle; and the sweep angle of the arc. The beginning angle is measured from the X-axis to the line joining the center and beginning points of the desired arc. The sweep angle is measured between the lines that connect the center of the arc to the beginning and ending points of the arc. (A counterclockwise angle is positive; a clockwise angle is negative.)

Figure 17 illustrates the following arc definitions:

```
A1 = ARC/ -1, 2, .8, 45, 90
A2 = ARC/ 1, 1.5, .6, 90, -135
A3 = ARC/ 1, 2.8, .9, 90, -90
A4 = ARC/ 3, 3.8, 1, 0, -90
A5 = ARC/ 4, 1, 1.5, 180, -180
```

ARC — Between Two Points

General Form:

```
                      XLARGE
ARC/ point, point,    XSMALL,  RADIUS, scalar, CLW
                      YLARGE
                      YSMALL
```

RADIUS is a language word that indicates the value of the radius is the next word. CLW is a language word that means clockwise.

An arc between two points can be defined when the relative position of the center and the value of the radius are known.

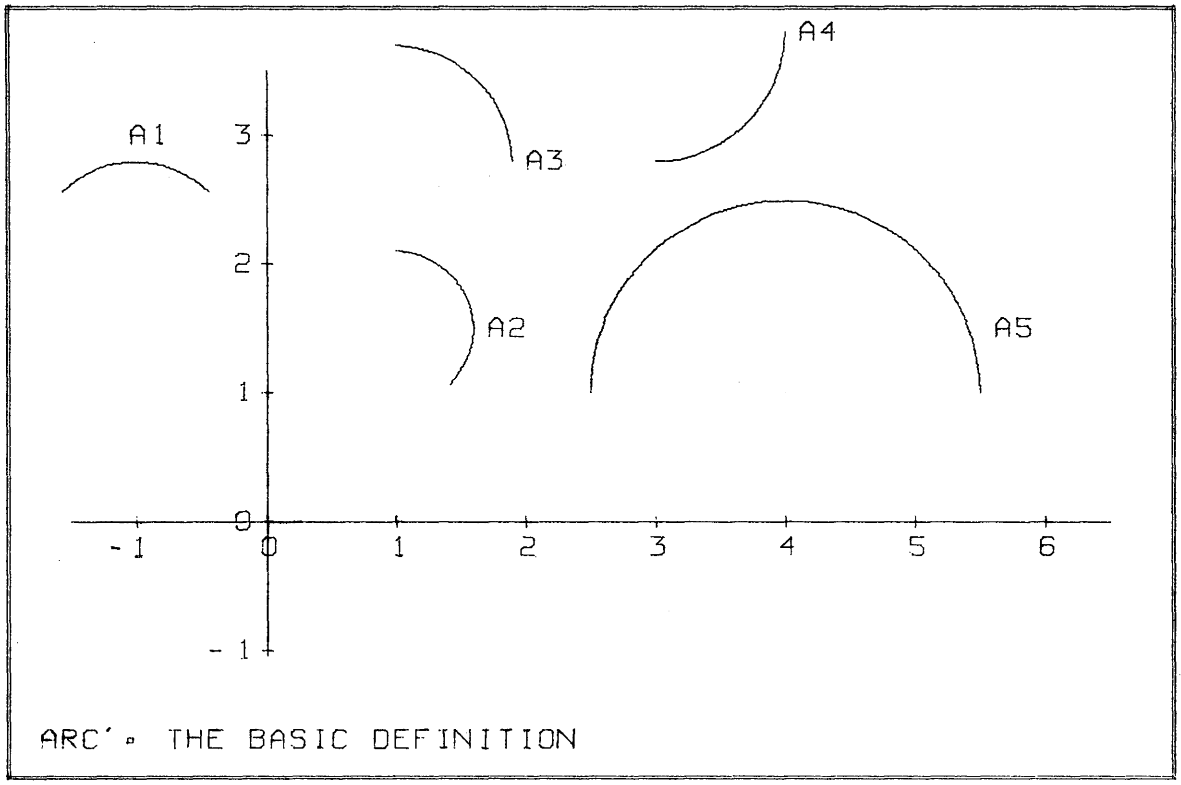


Figure 17

The four language words before RADIUS are used to select the proper arc by indicating the position of the center of the arc. For example, if the center of the arc is below the two end points, the proper word to use is YSMALL; if the center is to the left of the two end points, the proper word to use is XSMALL. When no word is specified, the processor assumes XLARGE.

The word CLW is optional in the arc definition. If the sweep of the desired arc from the first point to the second point is counterclockwise, no word is needed. When the sweep is clockwise, the word CLW is used.

When the radius of the arc is too small to pass through the two points, the processor types out an error message.

Coding examples for Figure 18:

A1 = ARC/P1, P2, XSMALL, RADIUS, 1.2, CLW

A2 = ARC/P3, P4, XLARGE, RADIUS, .75

A3 = ARC/L1, YSMALL, RADIUS, 2.1, CLW

A4 = ARC/L1, XSMALL, RADIUS, 2.1

A5 = ARC/ 2, 2.8, 3.5, 2.4, XSMALL, RADIUS, 1.4, CLW

A6 = ARC/P5, P6, YSMALL, RADIUS, .75, CLW

A7 = DOTTED, ARC/P5, P6, YLARGE, RADIUS, .75

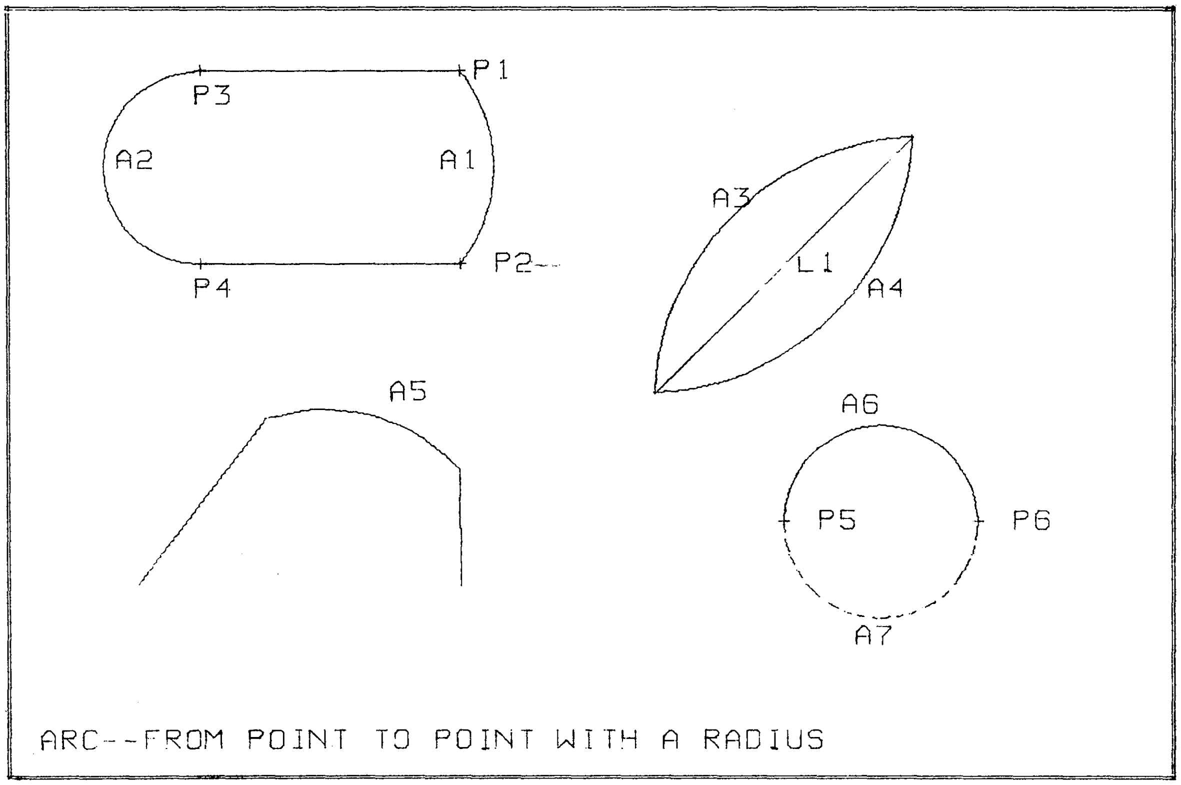


Figure 18

ARC — Tangent to Two Lines (Filleting)

General Form:

ARC/<u>scalar</u>

A filleting arc can be defined between any two nonparallel lines. The radius of the arc is indicated by the <u>scalar</u>. This statement is always preceded and followed by a line definition:

```
LINE/ Any line definition

ARC/scalar

LINE/ Any line definition
```

The two lines must be defined so that the ending point of the first line is the same as the beginning point of the second line. The line class for all three items — the two lines and the arc — is the line class of the second defined line.

The three statements cause the following operations: (1) an arc of the desired radius is constructed so that the arc is tangent to both of the lines, and (2) the two lines are shortened to coincide with the ending points of the arc.

The second line definition may be followed by another ARC statement and then another line definition. In such a case the three items are processed in the same manner as the first three statements. This provision enables the programmer to define a line which has fillet arcs on both ends. However, to produce a fillet on every corner of a closed figure, it is necessary to start with an intermediate point along one of the lines.

If a line definition in a filleting sequence is given a label, that label represents the full, unshortened line.

The following statements, depicted in Figure 19, show the operation of the fillet arc:

```
L1 = LINE/ .5, .5, DY, 2.5

ARC/1

L2 = LINE/DX, 3.5

ARC/1

L3 = LINE/PPP,6,1.5

ARC/.5

L4 = LINE/DY,-1.0

LINE/DX,-5.5
```

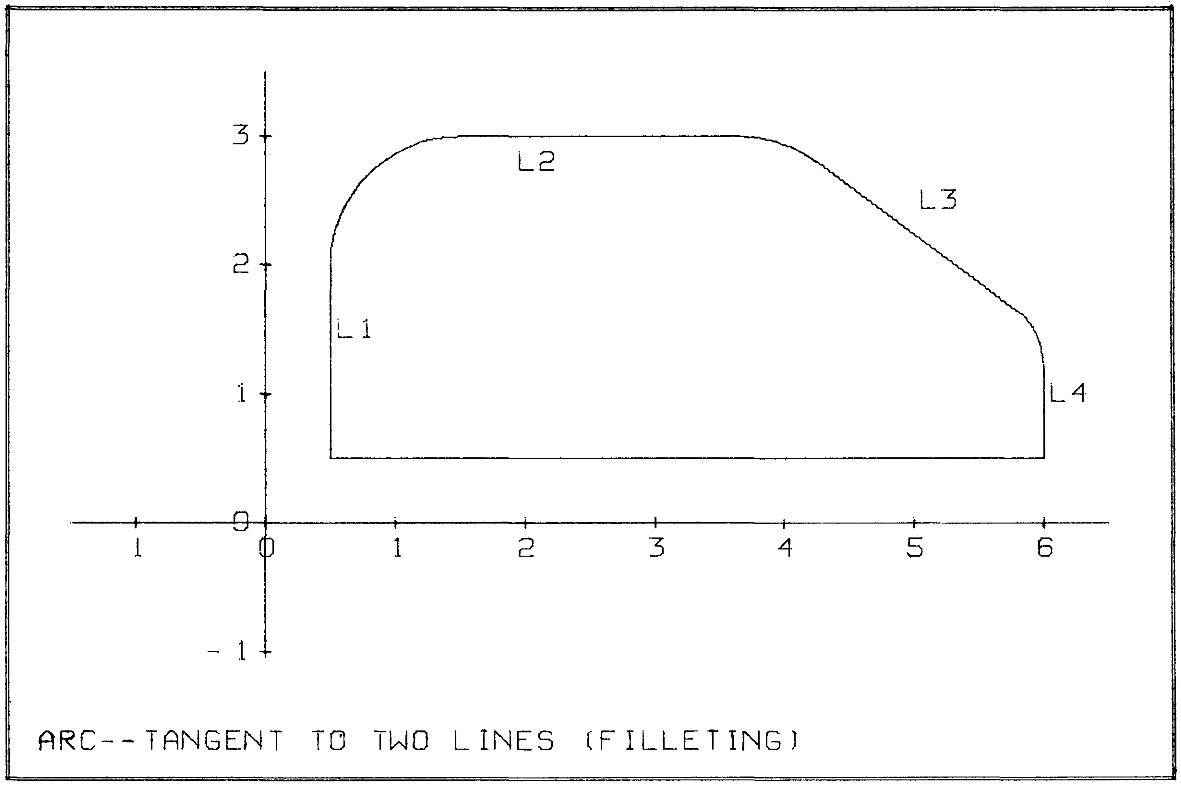


Figure 19

2.02.06 GEOMETRIC FUNCTIONS

Included in the drafting language is a group of function words that make it possible to operate upon geometric data. The geometric functions that can be performed are similar to arithmetic functions, but their arguments are always geometric. (The argument of a function is the element upon which the function operates and is always enclosed in parentheses.) The general form of a geometric function is the language word followed by the argument enclosed in parentheses. The following geometric functions are in the 1620 Drafting System:

Function	Purpose
DXOF (line)	The X-value of the first point on the line is subtracted from the X-value of the second point on the line. Example: N3=N1 * DXOF (LI) This statement sets N3 equal to the product of N1 and the value of the difference in X-values of the end points of the line labeled L1.

DYOF (line)

The Y-value of the first point on the line is subtracted from the Y-value of the second point on the line.

Example: POINT/4, DYOF (P1, P2)

DIST (line)

This function calculates the distance between two points or the length of a line.

Example: N4=DIST(LI) + DIST (P3, P4)

ATAND (line)

This function calculates the angle (in degrees) that a line makes with the horizontal axis. The calculation is performed by assuming that the first point of the line is the origin point. The answer is a positive angle in the range 0-359.9 degrees.

Example: N5 * ATAND (L14)

```
               point
PARAM (scalar, line)
              circle
                 arc
```

This function selects a parameter from a point, line, circle, or arc.

A POINT is defined with two parameters in this sequence:

(1) X of the point
(2) Y of the point

A LINE is defined with four parameters in this sequence:

(1) X of the first point
(2) Y of the first point
(3) X of the second point
(4) Y of the second point

A CIRCLE is defined with three parameters in this sequence:

(1) X of the center point
(2) Y of the center point
(3) the radius of the circle

An ARC is defined with five parameters in this sequence:

(1) X of the center point
(2) Y of the center point

(3) the radius
(4) the beginning angle
(5) the sweep angle

Suppose that the programmer wishes to make a statement which references the radius of the circle C2 that has been defined as:

```
C2=CIRCLE/P1, P2, P3
```

The value of the radius is not known by the programmer, but it can be obtained by writing:

```
R=PARAM (3, C2)
```

The 3 identifies the parameter desired, and C2 is the label of the circle from which the radius is taken.

Mirror Functions. The 1620 Drafting System enables a programmer to draw mirror images (see section 2.04.02). To place annotations on a mirror image, it is necessary to reference the mirrored value of points. The following three functions allow the programmer to define points that are the reflected values of other given points. These functions may be placed wherever a point definition is allowed.

Function	Purpose
MIRX (<u>point</u>)	Reverses the sign of the X-coordinate of <u>point</u>, thereby reflecting <u>point</u> about the vertical axis. Example: LINE/ MIRX (P1), MIRX (P2)
MIRY (<u>point</u>)	Reverses the sign of the Y-coordinate of <u>point</u>, thereby reflecting <u>point</u> about the horizontal axis. Example: DIM/ P1, MIRY (P1)
MIRXY (<u>point</u>)	Reverses the sign of the XY-coordinates of <u>point</u>, thereby reflecting <u>point</u> about the horizontal and vertical axes simultaneously; this has the effect of mirroring about a line that is -45 degrees to the X-axis when <u>point</u> lies in the first or third quadrant, and reflecting about a line +45 degrees to the X-axis when <u>point</u> lies in the second or fourth quadrant. Example: CIRCLE/MIRXY (P1), .5

2.02.07 PART COORDINATE SYSTEM

When describing an object in the Drafting System language, a programmer must make use of coordinates. Because the values used in definition statements are the actual dimensions of the part, an object is described in the Part Coordinate System. This coordinate system has a main part origin and may also have local coordinate systems.

1. The Part Origin. Before the view of an object can be coded in the language, a part origin must be established. The programmer may choose any point as the part origin. The best point to choose is the one from which most of the dimensions of the object can be defined easily. For example, the lower left-hand corner of a rectangular object is a natural choice. When defining a circular object, the programmer normally would choose the center of the object as the part origin.

   In addition to the origin, the orientation of the axes for the coordinate system must be established. In this case, also, it is best to choose an orientation that makes the dimensions of the object easily definable. For a rectangular object the natural orientation would be one in which the axes are parallel to the edges of the object.

   After the programmer has chosen a part origin (and an axis angle), he begins defining the points, lines, circles, and arcs that make up a view of the object. Each of these geometric elements may be defined by coordinates that are relative to the selected part origin. There are <u>no</u> language statements by which the programmer would define the part origin that has been chosen. For proper part description it is necessary only that the programmer be consistent when using coordinates.

2. Local Coordinate Systems. It is sometimes convenient to be able to establish a local coordinate system for an object. This is true particularly when part of the object is at an angle to the main body of the object. In such a case, the programmer selects a part origin suitable for the main body of the object, and then defines a local coordinate system (with respect to the main part origin) that facilitates coding the part of the object which is oblique. The REFSYS statement is used to define a local coordinate system.

General Forms:

REFSYS/ <u>point</u>	(Translation only)
REFSYS/ <u>point</u>, ATANGL, <u>scalar</u>	(Translation and rotation)
REFSYS/ ANGOF (<u>line</u>)	(Translation and rotation)
REFSYS/ NOMORE	(End REFSYS)

<u>point</u>	A point in the main part origin system that becomes the origin of the local coordinate system.
ATANGL	A language word which indicates "at an angle".
<u>scalar</u>	The angle in degrees which the new axis makes with the part origin axis.
ANGOF	A language word which indicates "angle of".

<u>line</u>	A line defined in the part origin system. The first point of the line becomes the origin of the local coordinate system. The angle of the new axis is the same as the angle of the line.
NOMORE	A language word that indicates "no more".

Figure 20 illustrates the effect of a local coordinate system. The main part origin is shown at (0, 0), and the axes are parallel to the edges of the paper. At the point (1, 3) a local coordinate system has been defined by the statement:

REFSYS/1, 3

This local coordinate system is an example of translation only. The second example in Figure 20 shows translation and rotation. At the point (8, 3) a local coordinate system has been defined that is rotated 45 degrees from the main part axis. Either of the following two statements defines this local system:

REFSYS/8, 3, ATANGL, 45

REFSYS/ ANGOF (8, 3, 9, 4)

After a REFSYS statement has been given, all part coordinates must be defined in this new coordinate system. In this respect, a REFSYS statement is modal, because it affects statements which follow the REFSYS definition. Specifically, a REFSYS statement affects all POINT, LINE, CIRCLE, ARC, NOTE, NOTER, and DIM statements. The effect of a REFSYS statement ends when (1) a new REFSYS statement is written, or (2) the statement REFSYS/NOMORE is written. When a new REFSYS statement is given, that statement affects all succeeding statements; when REFSYS/NOMORE is given, all coordinates in succeeding statements are with reference to the main part origin.

There is one restriction. When a REFSYS has been given, no filleting arcs (see section 2. 02. 05) may be defined.

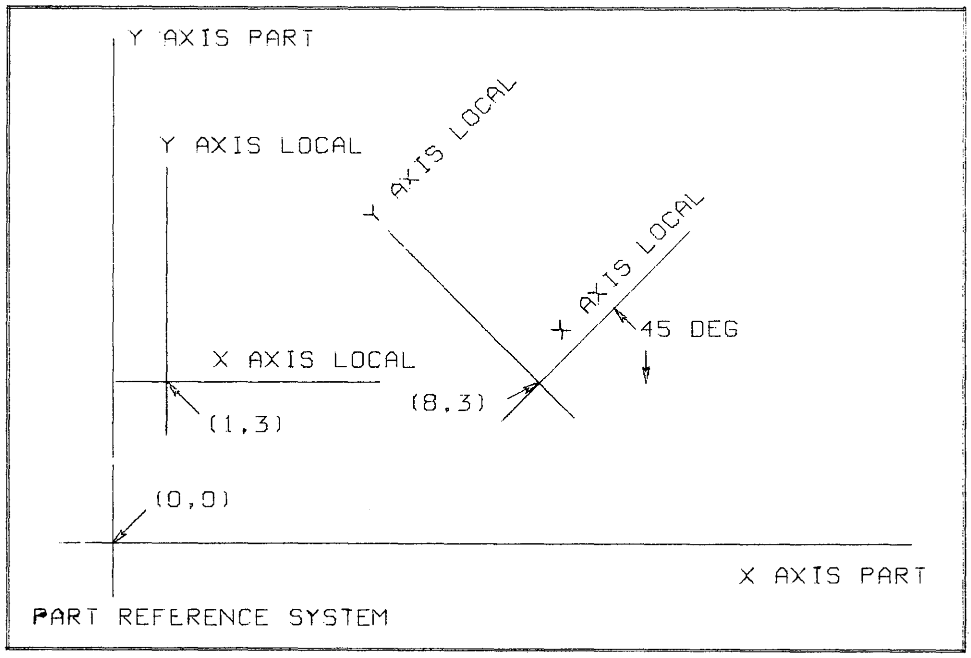


Figure 20

2.03 ARITHMETIC STATEMENTS

A numeric value needed to code a drawing is not always directly available. The desired number may be the resultant value of one or more arithmetic calculations. The arithmetic capabilities of the language allow such operations as the sum of two numbers, or the sine of an angle, to be performed by the 1620 Drafting System. Two types of language words are used — arithmetic operators and arithmetic functions.

2.03.01 ARITHMETIC OPERATORS

Addition +	A plus sign indicates that two numbers are to be added together: N1 + 4 3.67 + N2
Subtraction -	A minus sign is used to obtain the difference of two numbers: N3 - N2 N9 - 6.431
Multiplication *	The asterisk is used to cause multiplication: N2 * N4 3.14159 * N3
Division /	A slash is used to cause division. To perform the operation N1 divided by N2, write: N1/N2 If division by zero is attempted, an answer of 10^{99} is given by the drafting program.
Exponentiation **	Exponentiation is indicated by using two adjacent asterisks. For instance, to cube the number N1, write: N1 ** 3

2.03.02 ARITHMETIC FUNCTIONS

A numeric value, such as the cosine of an angle or the square root of a number, can be obtained in the 1620 Drafting System by using language words known as arithmetic functions. For example, the value of the cosine of an angle of 72 degrees can be used by writing:

```
COSD (72)
```

This example illustrates the general format for writing arithmetic functions — the function word, followed by the argument of the function within parentheses. The argument can be any number or any arithmetic expression that results in a numeric value. The following functions are in the 1620 Drafting System.

Sine

To take the sine of an angle in degrees, write:

SIND (scalar)

where scalar represents any number or arithmetic expression that results in a numeric value.

Cosine

The cosine of an angle in degrees is obtained by writing:

COSD (scalar)

Arctangent

To find the angle whose tangent is known, use the arc tangent function, and write:

ATAND (scalar)

The argument of this function is the value of the tangent, and the answer is an angle in the range -90 to 90 degrees.

Square Root

The square root of a number can be taken by writing:

SQRT (scalar)

If the argument is a negative value, the system takes the square root of the number as if the number were a positive value.

Absolute Value

The absolute value of a number can be taken by writing:

ABS (scalar)

Natural Logarithm

The natural logarithm of a number can be taken by writing:

ALOG (scalar)

An attempt to take the logarithm of a negative number results in the system using the absolute value of the argument. The logarithm of zero is -10^{99}.

The Exponential Function — The antilogarithm of a number in the natural system is obtained by writing:

EXP (<u>scalar</u>)

The arithmetic operators and functions can appear together in statements. The following examples illustrate valid phrases which can be written in any language statement where a numeric value is needed:

```
N1 + N2 - N3
SQRT (N1 * 7)
4.6 / ABS (N18)
N6 - ATAND (SIND (N5)/COSD (N5))
```

2.03.03 PARENTHESES IN ARITHMETIC EXPRESSIONS

The expression

N1 * N2 + N3

is evaluated by first multiplying N1 times N2, then adding N3. If the addition must take place first, write N1 * (N2 + N3). Parentheses are used in this manner to indicate the order in which operations are to occur.

The need for parentheses can be determined by knowing the order in which operations are performed if no parentheses are present.

The hierarchy of operations is:

First	-Exponentiation
Second	-Multiplication and division
Third	-Addition and subtraction

For example, the expression:

A + B * C ** D/E - F

in the drafting system is equivalent to the algebraic expression:

$$A + \frac{BC^D}{E} - F$$

while A + B * (C ** (D/E) - F) in the drafting language is equivalent to: $A + B\,(C^{D/E} - F)$ in algebraic notation.

2.03.04 USING ARITHMETIC EXPRESSIONS

An arithmetic expression can appear by itself in a statement:

```
N1 = N3 * N4 - SIND (45)
N9 = 0.01
```

or it can appear in any statement where a scalar value is needed:

```
POINT/N3 - 21, 7
CIRCLE/3.4, 9, 1.643* .5
ORIGIN/ 2 + N1 * COSD (N2), 2+N1 * SIND (N2)
```

2.04 DRAWING STATEMENTS

The programming of a drawing in the 1620 Drafting System is always done in two parts. First, the part to be drawn is described, and second, drawing commands are written that define what is to be drawn and how it is to be drawn. The preceding sections of the manual have presented the statements used to describe an object; this section explains the statements which cause drawing to be performed. The section is divided as follows:

1. SCALE and ORIGIN statements are explained in section 2.04.01. These statements set the scale factor for the drawing and place the object lines at a desired location on the paper.

2. The drawing action commands, presented in section 2.04.02, are used to operate upon views and shapes. The DRAW statement causes a set of object lines to be drawn. The HATCH statement causes crosshatch lines to be drawn through a view or shape. The way to draw a mirrored reflection of a view or shape is also explained in this section.

3. The way in which lettering is performed is discussed in section 2.04.03. The ALPHAP statement, which controls the size and spacing of characters, is explained.

4. The statements that cause both local and general notes are explained in section 2.04.04.

5. Dimension statements are explained in section 2.04.05. These statements are used to tell how to dimension, what to dimension, and how to display the dimension on the drawing.

2.04.01 PAPER COORDINATE SYSTEM

Before the view of an object can be drawn, it is necessary to write statements that define the scale factor for the drawing and the location on the paper where the view is to be drawn. These items are required because a part is described in its true dimensions with coordinates that reference the part origin coordinate system. To produce a drawing, the coordinates of all object lines must be converted to the paper coordinate system. This conversion is handled automatically by the 1620 Drafting System. The programmer need only (1) define the scale factor in a SCALE statement, and (2) define the location on the paper where the view is to be drawn by writing an ORIGIN statement.

1. The SCALE Statement. A SCALE statement can be written in any of the following forms:

   SCALE/<u>scalar</u>

   SCALE/<u>scalar</u>, <u>scalar</u>

   SCALE/NOMORE

   To produce a drawing with a double scale, for example, write: SCALE/2. A one-quarter-size drawing can be specified by writing: SCALE/ 1/4.

   It is sometimes desirable to have separate scale factors for X and Y dimensions. For example, to create a drawing with a scale factor of three on the X values and a scale factor of five on the Y values, write: SCALE/3, 5. (Note that circles and arcs are scaled by the Y scale factor only, so that circles are drawn round even when two different scale factors are defined.)

   The SCALE statement is modal; once a scale factor is defined, that factor affects all succeeding program statements which reference part coordinates. To terminate the effect of a SCALE statement, write (1) another SCALE statement that defines a new scale factor, or (2) the statement SCALE/NOMORE, which prevents scaling altogether and so causes true size representations.

2. The ORIGIN Statement. The general forms are:

ORIGIN/<u>point</u>	(Translation only)
ORIGIN/<u>point</u>, ATANGL, <u>scalar</u>	(Translation and rotation)
ORIGIN/ANGOF (<u>line</u>)	(Translation and rotation)
ORIGIN/NOMORE	(End ORIGIN)

ATANGL	A language word indicating that the value following it is an angle.
ANGOF	A language word that indicates rotation at the angle of the given line. The rotation occurs about the first point of the given line.
NOMORE	A language word that indicates "no more".

The ORIGIN statement is used to place the view of an object at a desired location on the paper. The desired location is written as a point in paper coordinates where the part origin is to be placed.

The paper coordinate system is defined in the following manner: The paper origin is the lower, left-hand corner of a drawing. The positive X-axis is the bottom edge of the paper. The positive Y-axis is the left edge of the paper. All drawing, therefore, is done in the first quadrant of this coordinate system.

Figure 21 illustrates the paper coordinate system. The paper origin is shown as the point (0, 0), and the two axes are identified as X AXIS PAPER and Y AXIS PAPER. At the point (1, 3), the scaled coordinate system for the part is shown. This translation can be accomplished by writing:

ORIGIN/1, 3

If a DRAW statement is written after the ORIGIN statement (such as DRAW/FRONT), the view labeled FRONT is drawn so that the part origin (see section 2. 02. 07) is located at the point (1, 3).

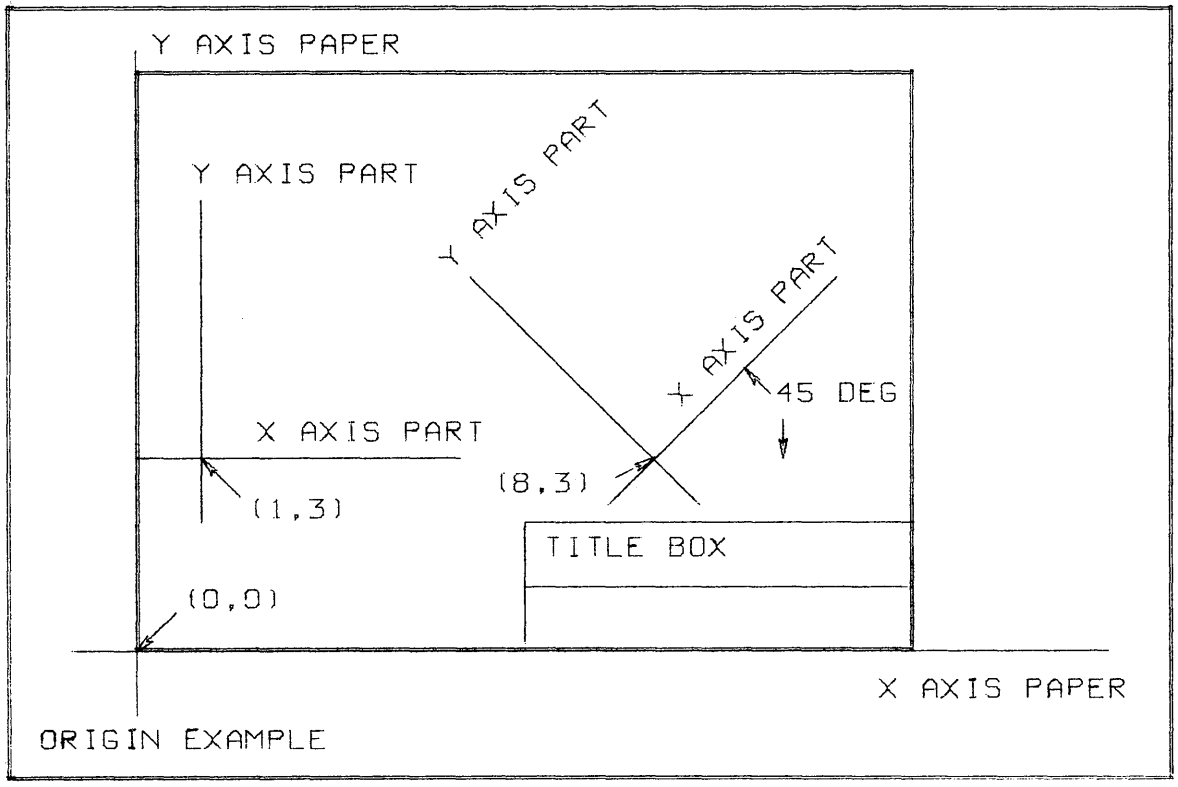


Figure 21

It is also possible to rotate the drawn view. At the point (8, 3) is an example of rotation which can be defined by either one of the following statements:

ORIGIN/8, 3, ATANGL, 45

ORIGIN/ ANGOF (8, 3, 9, 4)

The ORIGIN statement is modal; any drawing or annotation statements that follow the ORIGIN statement are placed on the paper according to the specifications in the ORIGIN statement. The specifications can be changed by writing another ORIGIN definition or by writing ORIGIN/NOMORE, which is equivalent to ORIGIN/0, 0.

The REFSYS statement (see section 2.02.07) and the ORIGIN statement should not be confused. To some extent it is possible to accomplish the same results with either statement. However, the two statements perform separate functions that should be clearly distinguished. The REFSYS statement is used to make it easier to describe a part; the ORIGIN statement is used to place a view on a sheet of paper. The REFSYS statement is used to convert a local coordinate into the main part coordinate system; the ORIGIN statement is used to convert a part coordinate into the paper coordinate system. In general, the REFSYS statement is used rarely, whereas the ORIGIN statement is used for every view.

Note that it is possible to have negative part coordinates, but that all paper coordinates must be positive. It is possible for an improper ORIGIN statement to cause negative paper coordinates; in such a case the 1620 Drafting System mirrors the points into positive values. Also, if points are developed with a Y-value greater than that which can be drawn on the paper, an error message is typed during processing. These checks are provided to prevent inadvertent plotting on previous drawings.

2.04.02 DRAWING COMMANDS

Drawing commands are used to draw object lines and crosshatch lines.

1. The DRAW Statement. The general form is:

$$\text{DRAW/ } \begin{array}{l}\underline{\text{view}}\\ \underline{\text{shape}}\\ \text{dim number}\end{array}$$

<u>view</u>	The label of geometric elements grouped together by a VIEW statement.
<u>shape</u>	The label of geometric elements grouped together by a SHAPE statement.
dim number	The 1620 Monitor dim number of a set of object lines stored on the disk pack.

The DRAW statement causes a previously defined set of object lines to be drawn. The size and placement of the drawn lines depend upon the most recent SCALE and ORIGIN statements. The usual sequence of language statements written to draw the view of an object is illustrated by the following example:

```
GEAR=VIEW/
          .
          .  Geometric definitions
          .
          END/GEAR
ORIGIN/4, 5
SCALE/.5
DRAW/GEAR
```

In this example, the statement DRAW/GEAR causes the drawing of all of the object lines defined in geometric statements that occur within the VIEW labeled GEAR. The SCALE statement causes the drawing to be done at one-half size. The ORIGIN statement causes the view to be located on the paper so that the part origin of the view is located at the point (4, 5).

The dim number form of the DRAW statement is used to draw a set of object lines that has been stored previously on the disk pack. This statement permits the drawing of templates and user-created graphic data. Templates are items (such as schematics of resistors or welding symbols) that are always drawn the same way, but that may differ in size and placement. An example of user-created data is the set of lines constituting the profile of a cam. The user may write a cam design program that outputs the profile of the cam onto the disk pack; the 1620 Drafting System can then be used to draw the profile. The specific details on how to store the object lines on the disk pack can be found in the System Manual for the 1620 Drafting System.

2. Crosshatching. Two statements are used: the HATCH statement, which draws the crosshatch lines, and the HATCHP statement, which tells how to draw the lines.

   The general form of the HATCH statement is:

```
HATCH/view
      shape
```

   This statement is used to draw crosshatch lines through the cross-sectional area that is defined by the object lines within a VIEW or SHAPE grouping. The normal use for this statement is illustrated at the bottom of Figure 22. The lines within a VIEW or SHAPE should form one or more closed envelopes. A closed envelope is a set of lines which enclose an area completely — such as a rectangle or a circle.

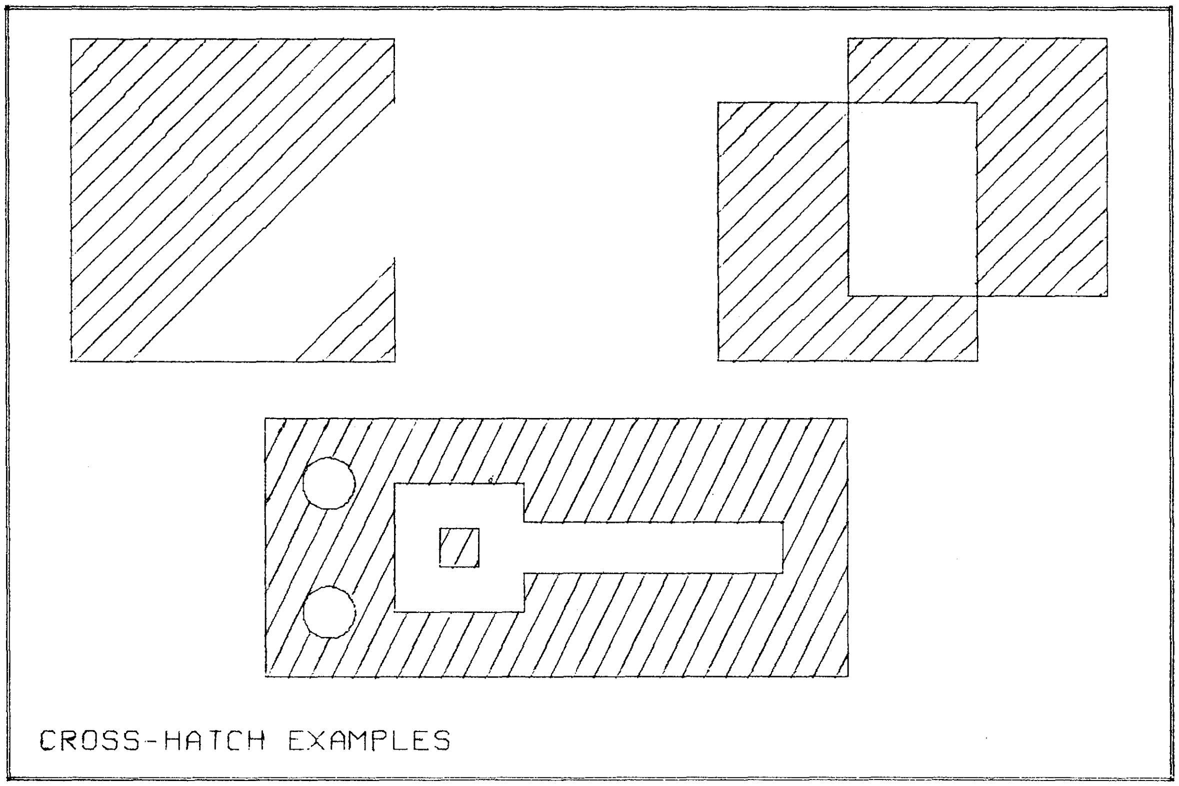


Figure 22

Crosshatch lines are developed in this manner: An infinitely long line is passed through every line within the geometric group. A set of points of intersection is calculated. These points are the intersections of the infinitely long crosshatch line with the lines in the geometric group. If there are no intersections, no crosshatch line is drawn; if there are two points of intersection with the crosshatch line, a crosshatch line is drawn between the two points. If there are more than two points of intersection, a crosshatch line is drawn between the first two points. No line is drawn between the second and third points. A line is drawn between the third and fourth points. And so the crosshatch line continues between every other set of points. The processor for the 1620 Drafting System then develops another infinitely long crosshatch line and restarts the calculations. When there are no more intersections to be made, the processor terminates the calculations.

An odd number of points of intersection is developed whenever there are incomplete figures. This condition is illustrated by the drawing in the top left portion of Figure 22. The drawing represents a square whose right edge is incomplete. In the area that is incomplete, only one point of intersection is found, and so no cross-hatch line is developed.

The output produced when two areas overlap is shown at the top right of Figure 22. No crosshatch lines are drawn within the overlapped area.

The HATCH statement, like the DRAW statement, is affected by the most recent SCALE and ORIGIN statements. The usual sequence of statements is illustrated in the following example:

```
SIDE=VIEW/
        .
        .  Geometric Definitions
        .
        END/SIDE
SCALE/.5
ORIGIN/2, 4
DRAW/SIDE
HATCH/SIDE
```

The format of crosshatch lines is controlled by the crosshatch parameter statement HATCHP. This statement has the general form:

line class, HATCHP/<u>scalar</u>, <u>scalar</u>, <u>scalar</u>, <u>scalar</u>

line class	Any of the language words for line class can be used, and causes all crosshatch lines to be of the defined class.
<u>scalar</u>	The four scalars are used to set the angle and spacing of the crosshatch lines. The first scalar is the angle that the crosshatch line is to make with the X-axis of the paper. The next three scalars are the distances between successive crosshatch lines.

The HATCHP statement is modal; once it is written, it affects the line format for all HATCH statements that follow. Figure 23 depicts some of the formats that can be drawn with the HATCHP statement. The examples show how the three distances are used to control the spacing of crosshatch lines. Patterns of one, two, or three lines can be drawn. A three-line pattern is drawn by specifying three distances; a two-line pattern is drawn by specifying two distances and a zero value; a one-line pattern is drawn by specifying a single distance and two zero values. If no HATCHP statement is written, the drawing format is that illustrated in the upper left of Figure 23.

3. Drawing Mirror Images. Three language words — MIRX, MIRY, MIRXY — are provided to make it possible to draw the reflected image of a view or shape. The image can be reflected about the horizontal axis, the vertical axis, or both axes simultaneously. The general forms for these statements are:

DRAW/MIRX $\left(\frac{\underline{\text{view}}}{\underline{\text{shape}}}\right)$	Reflect X-values
DRAW/MIRY $\left(\frac{\underline{\text{view}}}{\underline{\text{shape}}}\right)$	Reflect Y-values
DRAW/MIRXY $\left(\frac{\underline{\text{view}}}{\underline{\text{shape}}}\right)$	Reflect X- and Y-values

When a mirror image is drawn, the Drafting System processor reverses the sign of each coordinate in the view or shape and then applies the current SCALE and ORIGIN factors. MIRX causes the signs of the X-values to be reversed, thereby reflecting about the vertical axis; MIRY causes the signs of the Y-values to be reversed, thereby reflecting about the horizontal axis; MIRXY causes both signs to be reversed simultaneously.

Figure 24 shows how mirror images can be drawn. The small drawing in the upper right corner is the original coding of a VIEW labeled PLATE. The original is then redrawn with the three mirror images. The following coding produced Figure 24:

```
TITLE/ .3, .3, @DRAWING MIRROR EXAMPLE@
PLATE=VIEW/
LINE / 0, 2.8, .3, 2.8
LINE / PPP, .3, 2.6
LINE / DY, -1.5
LINE / DX, 2.5
LINE / DY, -.5
LINE / DX, -.5
LINE / DY, -.6
CTRLN, LINE / 4.5, 0, 0, 0
CTRLN, LINE / 0, 3.2, 0, 0,
CIRCLE / .75, 1.5, .3
LINE / 0, 1, 1, 0
END / PLATE
ORIGIN / 4.5, 3
SCALE / .8
DRAW / PLATE                    $$ FIRST QUADRANT
DRAW / MIRX (PLATE)             $$ SECOND QUADRANT
DRAW / MIRY (PLATE)             $$ FOURTH QUADRANT
DRAW / MIRXY (PLATE)            $$ THIRD QUADRANT
SCALE / .5
ORIGIN / 6.7, 4.3
DRAW / PLATE                    $$ ORIGINAL VIEW
FINI /
```

2.04.03 LETTERING

Lettering is placed on a drawing by writing note and dimension statements which contain the actual text to be drawn. The note and dimension statements are discussed in the next two sections of this manual. This section explains (1) the way in which a programmer can control the size and spacing of lettering, and (2) the set of characters that can be drawn.

1. Alphameric Parameters — control the size and spacing of lettering. The general form of the ALPHAP statement is:

$$\text{line class, ALPHAP/}(\underline{\text{scalar}},\ \underline{\text{scalar}},\ \underline{\text{scalar}},\ \underline{\text{scalar}}),$$
$$(\underline{\text{scalar}},\ \underline{\text{scalar}},\ \underline{\text{scalar}},\ \underline{\text{scalar}})$$

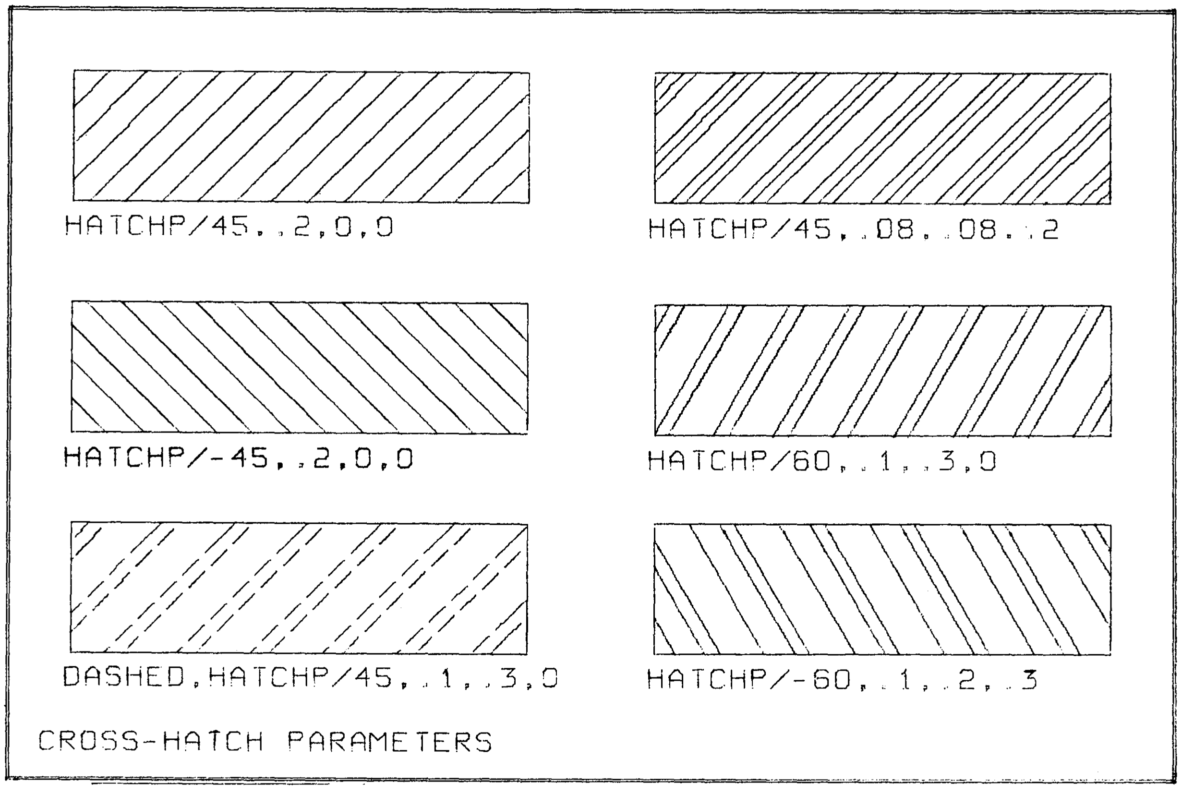


Figure 23

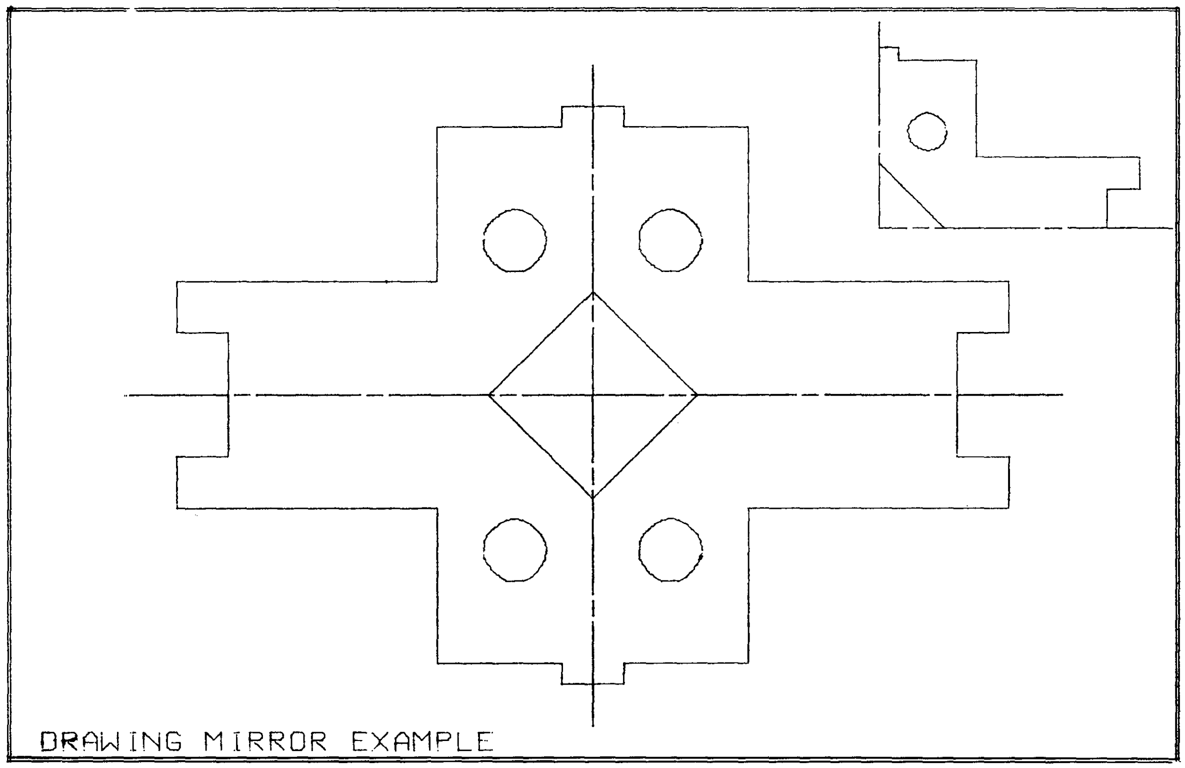


Figure 24

If a language word for line class is written, all lettering is done in that line class format. The eight scalars are used to define the size of the characters, the spacing between the characters, the spacing between the lines of text, the character set, and the angle of each letter. The following list identifies the purpose of each numeric value:

scalar 1	Font width.
scalar 2	Font height.
scalar 3	Horizontal spacing between the characters in a line of text.
scalar 4	Vertical spacing between the characters in a line of text.
scalar 5	Horizontal spacing between the first character of a line of text and the first character of the next line of text.
scalar 6	Vertical spacing between the first character of a line of text and the first character of the next line of text.
scalar 7	The character set, which is normally a zero. Other digits can be used to select user-created character sets.
scalar 8	The angle at which each character is to be drawn.

Figure 25 shows the meaning of each parameter. The letters A and B are drawn within boxes. The boxes represent the width and height of the font. A character is always drawn within the area of the font. The basic font size of the 1620 Drafting System is .1 by .1 inches. The actual character size within this font is .05 by .07 inches. Best results are obtained by using a font size which is a multiple of one-tenth of an inch. For example, a font size of .2 by .2 produces a character .1 by .14 inches. Figure 25 also shows that the horizontal and vertical spacings are measured between the centers of each font. The lower portion of the figure shows the way in which the character angle can be controlled.

Some sample lettering is shown in Figure 26. Below each example is drawn the statement which defined the alphameric parameters. The actual lettering is done with a note statement. The ALPHAP statement, therefore, is modal — it affects the action of succeeding statements in a program. Specifically, ALPHAP controls the way lettering is performed for a NOTE, NOTER, or TITLE statement. When no ALPHAP statement is written, the 1620 Drafting System uses the equivalent of this statement:

```
ALPHAP/(.2, .2, .16, 0), (0, -.2, 0, 0)
```

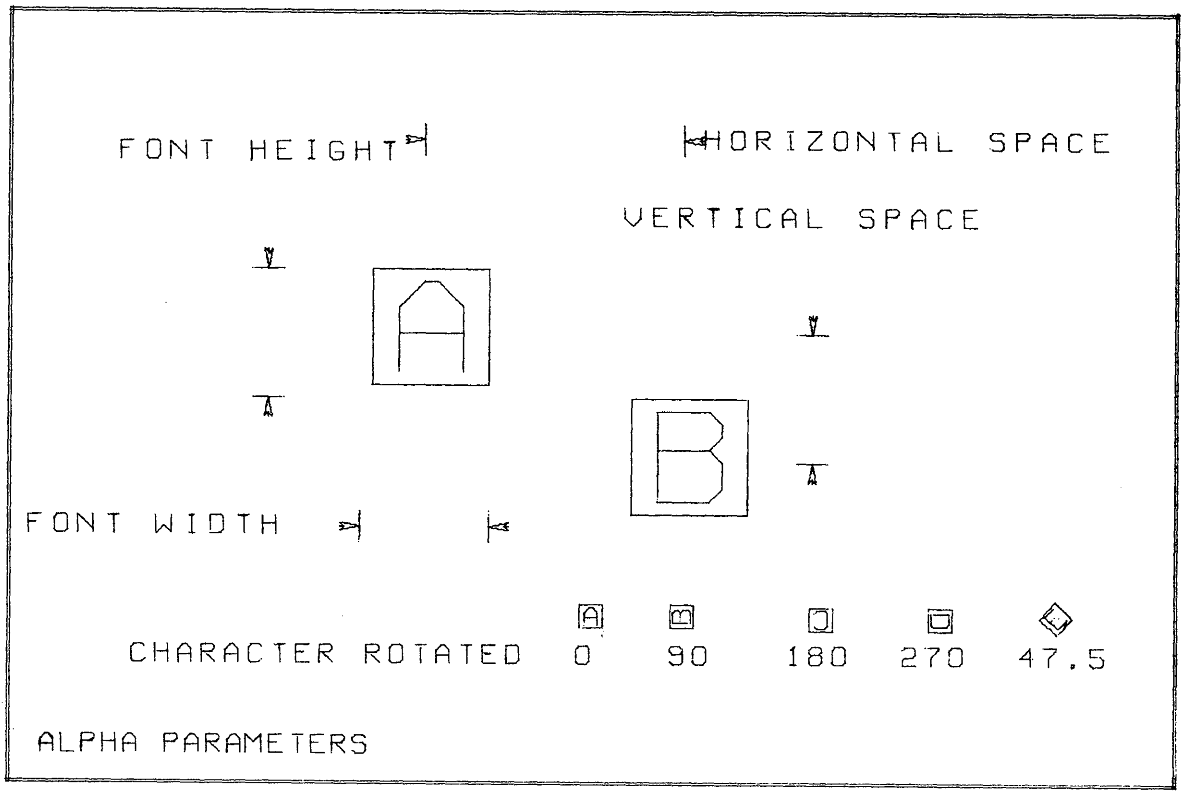


Figure 25

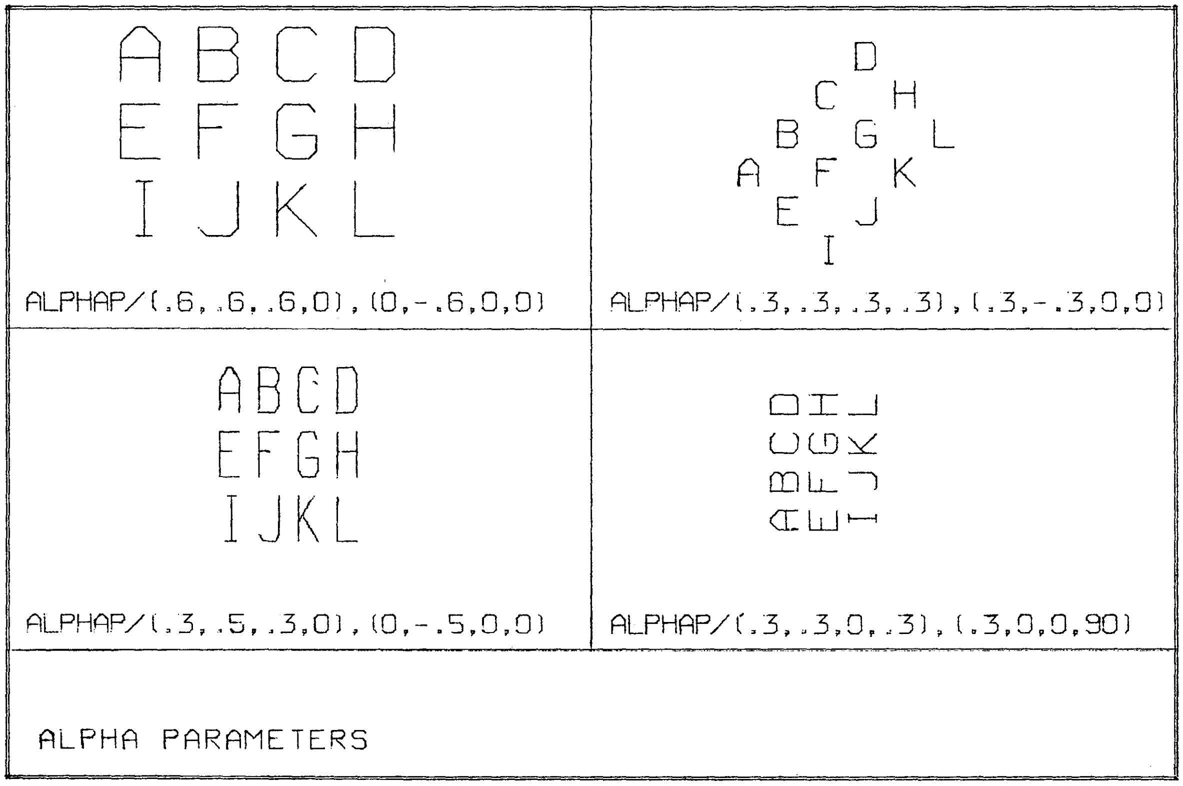

ALPHAP/(.6,.6,.6,0),(0,-.6,0,0)
ALPHAP/(.3,.3,.3,.3),(.3,-.3,0,0)
ALPHAP/(.3,.5,.3,0),(0,-.5,0,0)
ALPHAP/(.3,.3,0,.3),(.3,0,0,90)
ALPHA PARAMETERS

Figure 26

2. Character Set. A line of text can be placed on a drawing by a statement such as:

TITLE/2, 3, @PART 43@

TITLE is a language word used for drawing general notes. The location of the text on the drawing is given by the point (2,3). The actual text to be drawn is written as a literal — @PART 43@.

The top two lines of Figure 27 show the characters which can be written as part of a literal: the 26 alphabetic characters, the ten digits, and ten characters of punctuation. To draw any of these characters, it is necessary only to place the corresponding character in the literal of a note statement.

The 1620 Drafting System also enables the programmer to draw special symbols. These symbols are illustrated in the lower portion of Figure 27. To use one of these special symbols, the programmer writes, as part of the literal, a two-letter code, and encloses it with dollar signs. The special symbols provided are illustrated in Figure 27, with the graphic symbol first, the two-letter code, and the description of the symbol. To use the centerline character, the programmer would write CL.

Some of the symbols have special functions. BS causes a single backspace to allow the drawing of two characters in a single space. NL causes the characters following the code to be placed on a new line of text, according to the specifications of the current ALPHAP statement. BO causes a backspace, and then a box, to be drawn the size of the character font. US causes an underline to be drawn from that position back to the first character of the line.

Additional special symbols or a completely different character set may be included within the 1620 Drafting System. These different sets can be used by writing a digit other than zero in the ALPHAP statement. The method by which these character sets can be defined is discussed in the System Manual.

2.04.04 NOTE STATEMENTS

Note statements are action-type commands written to place local and general notes on a drawing. The language words used for these statements are NOTE, NOTER, and TITLE. These statements contain the text to be drawn and the desired location for the lettering. The ALPHAP statement is written to control the size and spacing of the drawn letters.

1. Local Note — a group of text associated with a point on the view of an object. The 1620 Drafting System permits three formats for local notes — text with an arrow from the left side, text with an arrow from the right side, and text with no arrow.

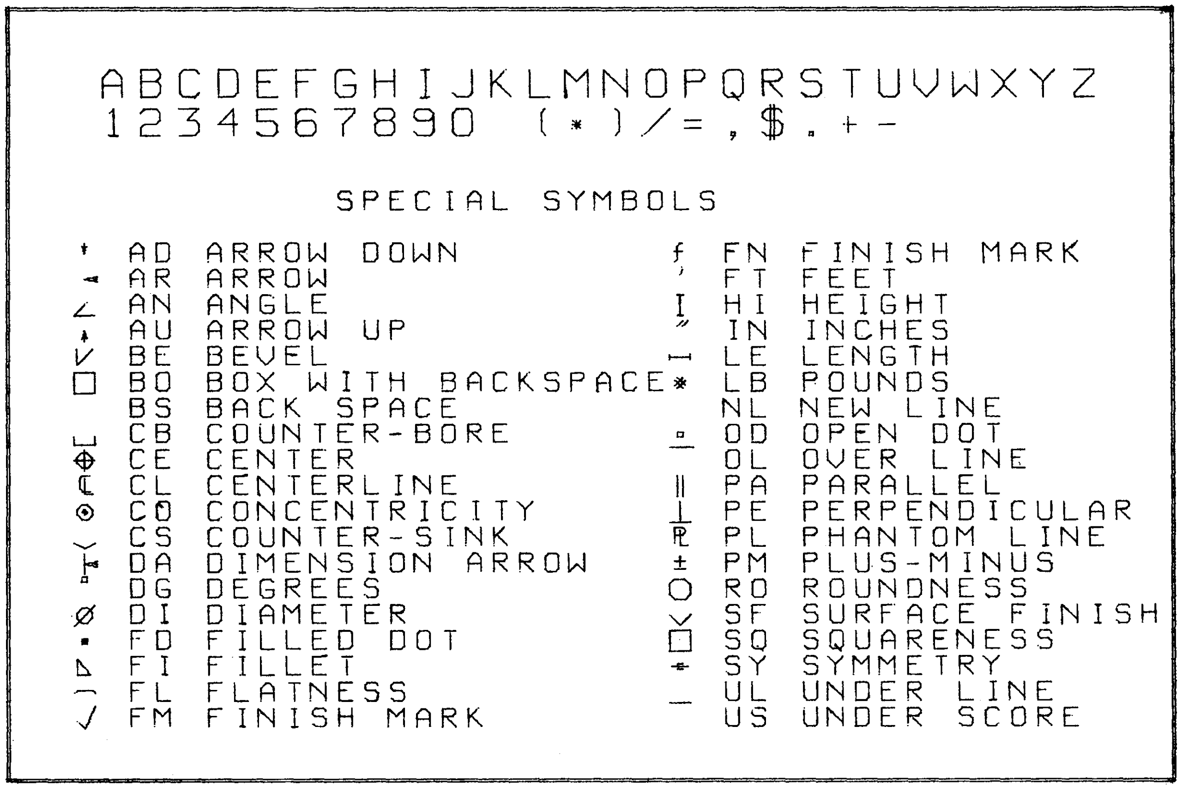
   The general form for a local note with an arrow drawn from the left is:

   NOTE/ $\underline{\text{point}}$, $\underline{\text{scalar}}$, $\underline{\text{scalar}}$, $\underline{\text{alpha}}$, . . .

NOTE	The language word for local notes.
<u>point</u>	The point on the view of the part to which the arrow points.
<u>scalar</u>	The horizontal distance (in paper inches) from the tip of the arrow to the end of the arrow.
<u>scalar</u>	The vertical distance from the tip of the arrow to the end of the arrow.
<u>alpha</u>,...	One or more literals (or labels of literals) which contain the text to be drawn.

This statement is used to draw one or more lines of text with an arrow from the first character of the first text line to the given point on the part. The two scalars affect the length of the arrow and, therefore, determine the location of the text.

Because the given point is a part location, the actual position on the paper is affected by the current values, if any, of SCALE, ORIGIN, and REFSYS. This feature of the statement enables the programmer to specify the position of notes in relation to the part without regard for the actual paper location.



ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890 (*)/=,$.+-

SPECIAL SYMBOLS

AD	ARROW DOWN	FN	FINISH MARK
AR	ARROW	FT	FEET
AN	ANGLE	HI	HEIGHT
AU	ARROW UP	IN	INCHES
BE	BEVEL	LE	LENGTH
BO	BOX WITH BACKSPACE	LB	POUNDS
BS	BACK SPACE	NL	NEW LINE
CB	COUNTER-BORE	OD	OPEN DOT
CE	CENTER	OL	OVER LINE
CL	CENTERLINE	PA	PARALLEL
CO	CONCENTRICITY	PE	PERPENDICULAR
CS	COUNTER-SINK	PL	PHANTOM LINE
DA	DIMENSION ARROW	PM	PLUS-MINUS
DG	DEGREES	RO	ROUNDNESS
DI	DIAMETER	SF	SURFACE FINISH
FD	FILLED DOT	SQ	SQUARENESS
FI	FILLET	SY	SYMMETRY
FL	FLATNESS	UL	UNDER LINE
FM	FINISH MARK	US	UNDER SCORE

Figure 27

When more than one literal is written, each literal is drawn as a new line of text. The spacing between lines is controlled by the specifications in the ALPHAP statement.

Figure 28 shows a local note with an arrow drawn from the left of the text. The point (0,0) is the part origin. The statement used to draw the note is:

```
NOTE/ .5, 2, .7, 1.0, @SAMPLE OF@, @NOTE, LEADER@, $

  @ON THE LEFT@, @NOTE/@
```

The following is the general form for a local note with an arrow drawn from the right:

```
NOTER/ point, scalar, scalar, alpha, . . .
```

This statement is used to draw one or more lines of text with an arrow from the last character of the first text line to the given point on the part. If the last character of the text is a blank, the arrow terminates after the blank character. Except for the position of the arrow, this statement operates in the same manner as the NOTE statement. Figure 28 shows a local note with an arrow drawn from the right of the text. The following statements were written to draw the note:

```
P1=POINT/6, 1.5

    NOTER/P1, -1.2, .6, @SAMPLE OF@,$

    @NOTE, LEADER@,$

    @ON THE RIGHT@,@NOTER/@
```

The statement to draw a local note with no arrow has the form:

```
NOTE/point, alpha, . . .
```

This statement is written in the same manner as the NOTE statement with an arrow from the left, except that the two scalars are omitted. The first character of the first line of text is placed on the paper with the font center at the given point. Figure 28 contains two examples of this statement. The language statements to produce these examples are:

```
NOTE/ 1, 1, @SAMPLE OF@, @NOTE WITHOUT@, @A LEADER@

NOTE/ 4, 1, @$CL$ CENTER LINE@
```

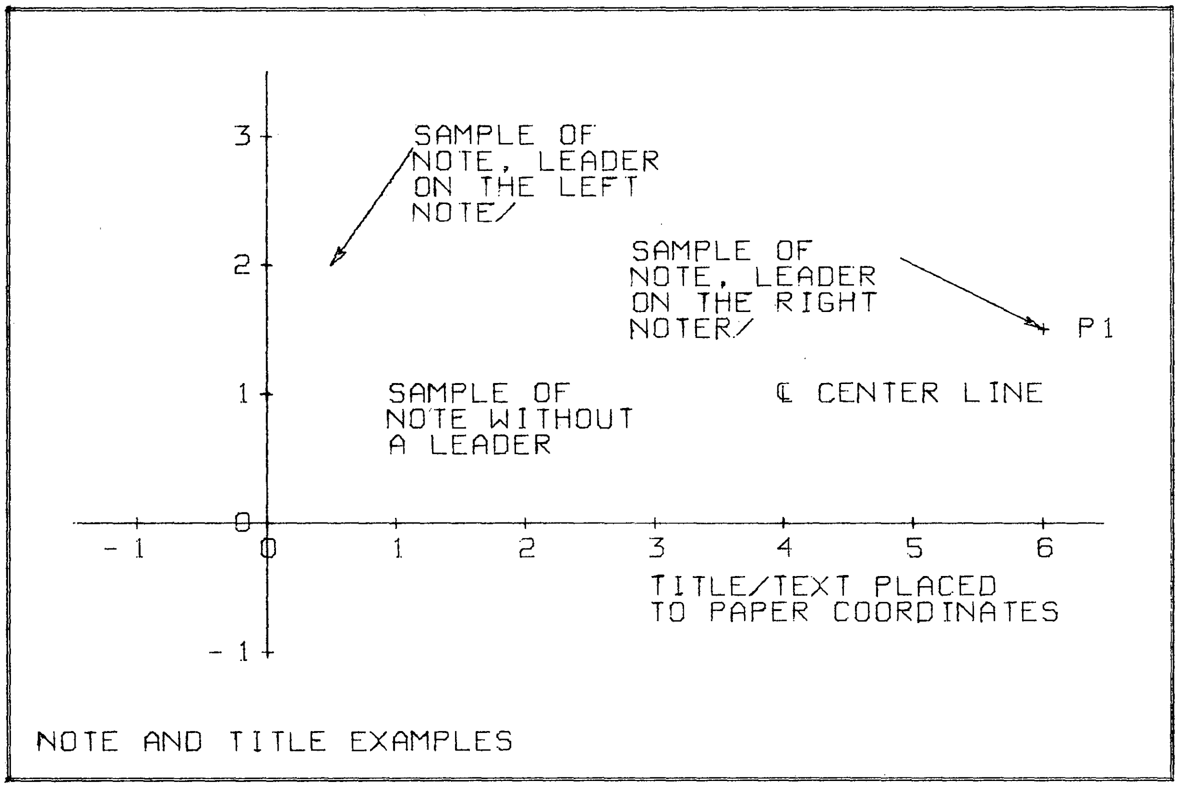

3
2
1
0
-1
-1
0
1
2
3
4
5
6
SAMPLE OF NOTE, LEADER ON THE LEFT NOTE/
SAMPLE OF NOTE, LEADER ON THE RIGHT NOTER/
P1
SAMPLE OF NOTE WITHOUT A LEADER
℄ CENTER LINE
TITLE/TEXT PLACED TO PAPER COORDINATES
NOTE AND TITLE EXAMPLES

Figure 28

2. General Note — a group of text placed at some convenient location on the drawing. The statement for drawing a general note is:

TITLE/<u>point</u>, <u>alpha</u>, . . .

TITLE	The language word used for a general note statement.
<u>point</u>	A point defined in paper coordinates at which the center of the first character is drawn.
<u>alpha</u>, . . .	One or more literals (or labels of literals) containing the text to be drawn.

The difference between the TITLE statement and a NOTE statement with no arrow is that the point given in the TITLE statement is expressed in paper coordinates. The TITLE statement can be used, for example, to place text in a title block on a drawing. Figure 28 depicts the following examples:

```
TITLE/ .3, .3, @NOTE AND TITLE EXAMPLE@
TITLE/ 5, 1.5,$
@TITLE/ TEXT PLACED@, $
@TO PAPER COORDINATES@
```

2.04.05 DIMENSION STATEMENTS

The dimension statements cause dimensions to be drawn in a variety of formats. To program dimensioning, these questions must be answered: (1) What is to be dimensioned? (2) Should dimensioning be by component or by true dimensions? (3) Where should the dimension line be placed? (4) How should the manufacturing tolerance be displayed? (5) What should the dimension line itself look like?

The drafting language provides flexibility with the following major words:

DIMST	Where and how to start dimensioning.
INDEX	Where to place the next dimension line in relation to the last dimension line.
DIMP	How to place the arrows and text for the dimension.
MASK	How to display the dimension value.
DIMxx	What to dimension.

DIMST — Where and how to start dimensioning.

General Form:

```
      XLARGE XCOMP
DIMST/XSMALL, YCOMP, line, scalar
      YLARGE TRUE
      YSMALL
```

This general form is equivalent to:

DIMST/modifier, type, starting line, indexing distance

Actual dimension lines are drawn by using a DIM statement which indicates the two points to be dimensioned. However, before a DIM statement can be written, a DIMST statement must be given to specify (1) whether the dimensions are to be component or true, and (2) where the dimension lines are to be placed. Once a DIMST statement is given, it affects all succeeding DIM statements.

The type of dimensioning is specified by one of the following:

XCOMP	All dimensions are horizontal component.
YCOMP	All dimensions are vertical component.
TRUE	All dimensions are true distance.

All dimension lines drawn after the DIMST statement are drawn parallel to the starting line given in the DIMST statement. The distance between the dimension lines is controlled by the scalar for indexing distance. The 1620 Drafting System spaces automatically between dimension lines. The direction of the spacing is controlled by the modifier in the DIMST statement.

The starting line is any line on the view being dimensioned. The line should be oriented the same as the desired dimensioning — for example, the line should be horizontal when XCOMP is used. The starting line may be a line that is going to be dimensioned in a subsequent DIM statement.

The indexing distance is a value expressed in paper inches. After a DIMST statement, the first dimension line to be drawn is drawn parallel to the starting line at a distance given by this indexing value. Subsequent dimension lines are also separated by this distance. The indexing distance is optional; a value of .5 is used when no previous value has been given. Once the programmer specifies a distance, that value is used until a new value is entered in a DIMST statement.

The modifier controls the direction of indexing. When YLARGE is written, automatic indexing occurs in the positive Y-direction; when XSMALL is written, the indexing occurs in the negative X-direction. If no modifier is written, the 1620 Drafting System assumes XLARGE.

An example of the DIMST statement is:

```
DIMST/XLARGE, YCOMP, L1, .7
```

In this case, L1 is a vertical line on the view of an object. YCOMP indicates that vertical component dimensioning is desired. All subsequent DIM statements cause dimension lines to be drawn parallel to L1. XLARGE indicates that the dimension lines are to be placed to the right of L1. The distance between dimension lines is indicated by the value .7.

Another example of the DIMST statement is:

```
DIMST/YSMALL, TRUE, P1, P2
```

In this case, all dimensioning is done parallel to the line formed by the points P1 and P2. No indexing distance is given, so a value of .5 is used. Each dimension line is indexed in the direction of smaller Y-values. The word TRUE indicates that true dimensions are desired.

Since all dimension lines are placed on the drawing at positions relative to the starting line in the DIMST statement, the actual location on the paper of the dimension lines depends upon the coordinates of the starting line. The starting line is defined in part coordinates, so that its location on the paper is affected by the current values, if any, of SCALE, ORIGIN, and REFSYS.

INDEX — Change the Location of Dimension Lines.

General Form:

INDEX/scalar

The INDEX statement allows the programmer to control the positioning of dimension lines. Normally, the 1620 Drafting System spaces automatically between dimension lines. By using an INDEX statement, the programmer can suppress the spacing or cause extra spaces to be skipped. The scalar in the statement is a positive or a negative number indicating the number of index spaces that the programmer desires to skip. The actual paper distance involved is equal to the product of the given scalar and the indexing distance entered in the DIMST statement.

To illustrate, consider these two dimension statements:

```
DIM/P1, P2

DIM/P2, P3
```

Each statement causes a dimension line to be drawn; the second line is spaced away from the first line. (The direction and distance of the spacing depend upon the DIMST statement.) By using an INDEX statement, the programmer can force the two dimensions to be drawn on the same line:

```
DIM/P1, P2

INDEX/ -1

DIM/P2, P3
```

The index value of minus one causes the second dimension to be placed on the same line as the first dimension. By using a positive index value, extra spaces can be placed between dimension lines. If a positive one were written in the preceding example, a space would be skipped, and the second dimension line would be placed two spaces away from the first line.

DIMP — Dimension Parameters.

General Form:

DIMP/scalar, scalar, scalar, scalar

The DIMP statement controls the placement of text and arrows for dimensioning. The general form of the statement is equivalent to:

DIMP/text placement, text size, "from" arrow, "to" arrow

The drawing of a dimension consists of a "from" arrow, a "to" arrow, and a block of text. Usually the text is centered between the two arrows that point out toward the points being dimensioned. The DIMP statement allows the programmer to control the format of presentation.

When the text is placed at the tip of the "from" arrow, the text placement factor is defined as zero; when the text is placed at the tip of the "to" arrow, the placement factor is defined as one. Therefore, to place the text in the center of the dimension line, a factor of .5 is specified in the DIMP statement. A factor greater than one causes the text to be placed beyond the "to" arrow. A factor of less than zero causes the text to be placed before the "from" arrow.

Figure 29 illustrates the results of various text placement factors. The value for text size determines the size of each character drawn. The number that the programmer specifies is the font size, which is somewhat larger than the actual character size. The basic font size for the 1620 Drafting System is .2 by .2 inches. This font produces a character .10 by .14 inches. For best results, the programmer should always specify a font size that is a multiple of the basic one-tenth of an inch. For example, a text size of .1 produces letters that are .05 inch in width by .07 inch in height.

The last two values in a DIMP statement are codes for the placement of the arrows. The codes are used in the following manner: A one indicates that a normal arrow is to be drawn; a zero indicates that no arrow is to be drawn; a two indicates that an opposite arrow is to be drawn. Normal and opposite arrows are defined with respect to the center point between the two dimension points. A normal arrow is directed from the center point toward the dimension point. An opposite arrow points toward the center point from a location beyond the dimension point. The examples in Figure 29 show how arrows can be controlled. The first four examples depict normal arrows; the next three show how one or both arrows can be omitted; the last three illustrate opposite arrows.

The DIMP statement is modal in that it affects all subsequent DIM statements. If the programmer does not specify a DIMP statement, the 1620 Drafting System uses the equivalent of the statement: DIMP/.5, .2, 1, 1.

MASK — Format the Dimension Text.

General Forms:

MASK/<u>alpha</u>

MASK/<u>scalar</u>, <u>scalar</u>, <u>alpha</u>

The MASK statement controls the format of the text which is displayed for a dimension. The general form of the statement is equivalent to:

MASK/manufacturing tolerances, display codes

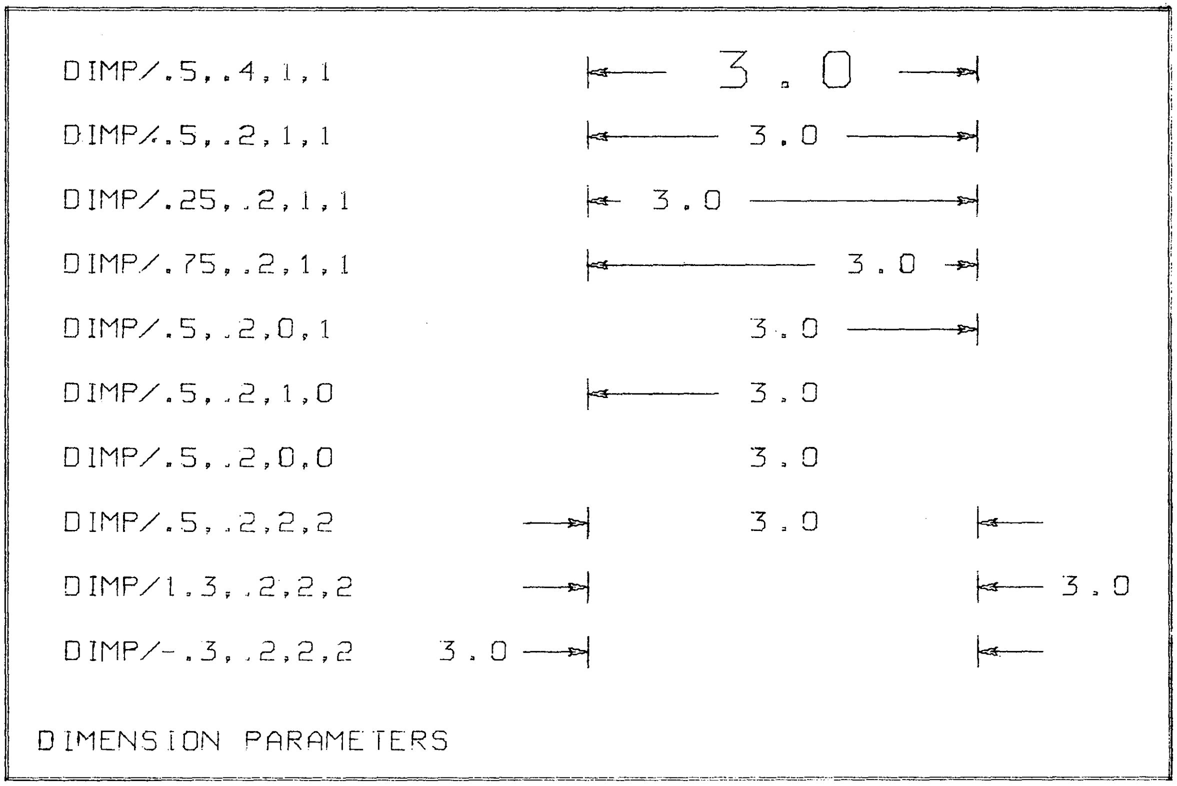


Figure 29

The two scalars represent an upper and lower tolerance. Either value may be minus. For example, if a dimension has a tolerance of plus or minus .001, write:

MASK/.001, -.001, display codes

For an upper tolerance of zero and a lower tolerance of minus .05, write:

MASK/0, -.05, display codes

The two scalars supply only the values for the tolerances; the display codes control the way in which the tolerances are to be drawn. The tolerances are written only if there is a display code for tolerance data.

The display codes are written within a literal. Three types of codes are used to (1) orient the text, (2) display decimal or fractional numbers, and (3) place tolerance information.

The codes used to orient the text are:

H	When this code is used, all dimension text is horizontal.

P	This code causes the lettering to be parallel to the dimension start line. The text is oriented so that it can be read from the bottom or the right side of the drawing.

The codes that control decimal or fractional display are:

Dd	This code causes decimal display. In place of the d, the programmer writes one of the digits 0 through 4. The digit specifies the number of decimal positions to be drawn. For example, to display 34.642, write D3.
Fdd	This code causes fractions to be displayed. In place of the dd, the programmer writes the base denominator for the fraction. For example, F08 displays fractions in eights; F64 displays fractions in sixty-fourths. The 1620 Drafting System converts a fraction to its lowest terms automatically. For example, a value of 4/32 is displayed as 1/8. If the fraction is zero, the 1620 Drafting System switches to decimal display, using the last value given in a D code.

The codes which place tolerance information in the text are:

TT	This code causes the tolerance values to be placed above the calculated dimension value.
TB	This code causes the tolerance values to be placed below the calculated dimension value.
TL	This code causes the calculated value to be displayed as a limit dimension. Two numbers are displayed: (1) the sum of the calculated value and the upper tolerance, and (2) the sum of the calculated value and the lower tolerance.
TN	This code indicates that no tolerance is to be displayed.

An example of the MASK statement with all three kinds of codes is:

```
MASK/ @HD3TN@
```

This statement causes dimension text to be displayed so that (1) the text is horizontal, (2) three decimal places are written, and (3) no tolerance is displayed. Blanks or commas can be used to separate the code fields. The following two statements are equivalent to the preceding example:

```
MASK/ @H D3 TN@

MASK/ @H,D3,TN@
```

The programmer does not have to write all three codes in every MASK statement. If a code is omitted, the 1620 Drafting System uses the last value for that code given in a preceding MASK statement. The specifications written in a MASK statement affect the display of dimensions caused by all subsequent DIM statements. If no MASK statement is given, the 1620 Drafting System uses the equivalent of:

MASK/ @P D0 TN@

Figure 30 illustrates various MASK statements. The MASK is used only when displaying calculated dimensions. The example on the left shows a special kind of dimensioning, in which the MASK statement is not used; the programmer has specified the exact text to be displayed:

DIM/P1, P2, @HARDEN@

The other examples in Figure 30 depict dimension statements of the form:

DIM/P1, P2

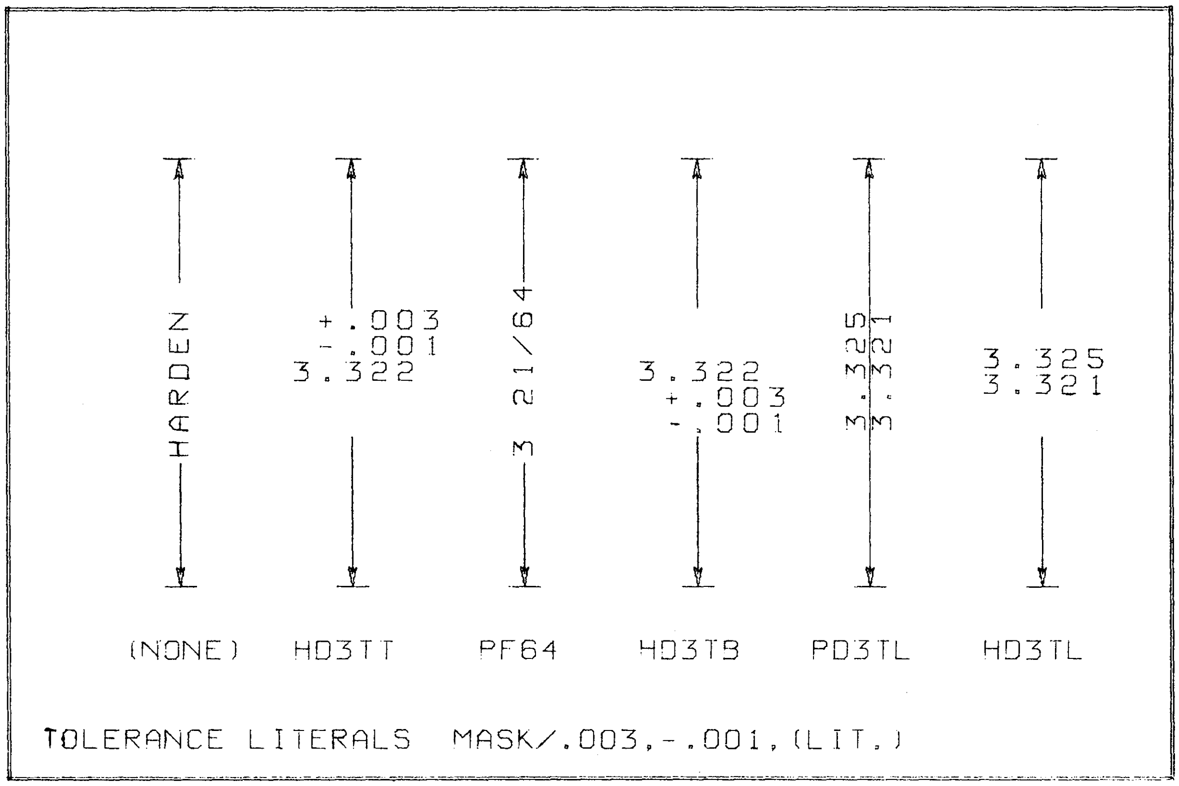


Figure 30

DIM — Draw a Dimension Line.

General Forms:

DIM/<u>line</u>,<u>alpha</u>

DIM/<u>line</u>,<u>point2</u>,<u>alpha</u>

DIM/<u>line</u>,<u>line2</u>,<u>alpha</u>

The DIM statement is an action-type command written to cause the drawing of dimension lines. A DIM statement causes:

1. Two points to be selected for dimensioning. When only a line is written in the DIM statement, the end points of the line are chosen. The first point of the line is called the "from" point; the second point of the line is called the "to" point. When a line and a point are written, the given point is the "to" point; the "from" point is the end point of the line that is closest to the location at which the dimension is to be drawn. When two lines are written, an end point is selected from each line.

2. The distance between the two points to be calculated. This distance is the value that is to be displayed on the drawing. However, if a literal is written in the DIM statement, the distance is not calculated, and the literal is displayed on the drawing.

3. The location of the dimension line to be determined. Two points are selected, so that the line between them is parallel to the starting line in the DIMST statement. The two points are the projections of the "from" and "to" points.

4. Extension lines to be drawn. These lines begin at the points to be dimensioned and extend to the location at which the dimension is to be placed.

5. Arrows to be drawn. The orientation of each arrow is specified in a previous DIMP statement.

6. Text to be drawn. The size and location of the text is determined by the values in a previous DIMP statement. The format of the text is controlled by the values given in a previous MASK statement.

   The language word DIM has four alternate forms. Each form causes the same actions as DIM, except for the drawing of extension lines.

DIMEE	Two extension lines are drawn (this is the same as DIM).
DIMNN	No extension lines are drawn.
DIMEN	An extension line is drawn only at the "from" point.
DIMNE	An extension line is drawn only at the "to" point.

Examples of dimension commands are:

```
DIM/L1

DIM/P1, P2

DIMEN/L1, @GRIND@

DIM/L1, P3

DIM/L1, L2, @A@
```

Figure 31 illustrates dimension lines. The drawing is produced by the following program:

```
TRI=VIEW/
L1 =LINE/DX,4           $$BOTTOM LINE
L2 =LINE/DY,2.8         $$RIGHT EDGE
L3 =LINE/PPP, 0, 0,     $$OBLIQUE LINE
P1 =POINT/3.1,.99       $$CENTER OF THE CIRCLE
C1 =CIRCLE/P1,.5        $$CIRCLE WITH .5 RADIUS
    END/TRI
$$END OF PART DESCRIPTION
$$
ORIGIN/2,2
$$
$$BEGIN DIMENSION STATEMENTS
$$
MASK/@P D2 TN@          $$P — PARALLEL TEXT, D2 — TWO PLACE DECIMALS
                        $$TN — NO TOLERANCES
DIMP/.5, .2, 1, 1       $$.5 — PLACE TEXT IN CENTER, .2 — CHARACTER
                                                            SIZE,
                        $$ 1, 1 — NORMAL ARROWS
DIMST/YSMALL,XCOMP,L1, .5
                        $$BEGIN AT LINE L1. HORIZONTAL DIMENSIONS.
                        $$INDEX DOWN .5 INCHES
DIMNE/(POINT/L3),P1
                        $$ 3.10
DIM/L1                  $$ 4.00 THIS DIMENSION IS AUTOMATICALLY
                        $$INDEXED BELOW THE PREVIOUS ONE
$$
DIMST/XLARGE,YCOMP,L2
                        $$BEGIN AT RIGHT EDGE. VERTICAL DIMENSIONS
                        $$INDEX TO THE RIGHT
DIMNE/L1,P1             $$.99
INDEX/-1                $$NEXT DIMENSION WILL BE ON SAME LINE
```

```
DIMNN/P1,POINT/L2
                      $$ 1.81
DIM/L2, @TEXT@        $$USE LITERAL TEXT
$$
DIMST/YLARGE,TRUE,L3
                      $$BEGIN TRUE DIMENSIONS FROM L3
DIM/L3                $$ 4.88
```

DIMCR — Dimension a Circle

General Form:

DIMCR/ circle, scalar, alpha

circle	The circle to be dimensioned.
scalar	The angle in degrees at which the dimension is to be drawn. The angle is measured from the bottom edge of the drawing paper.
alpha	Optional text to be drawn at the dimension line. If no text is given, the value of the radius is drawn.

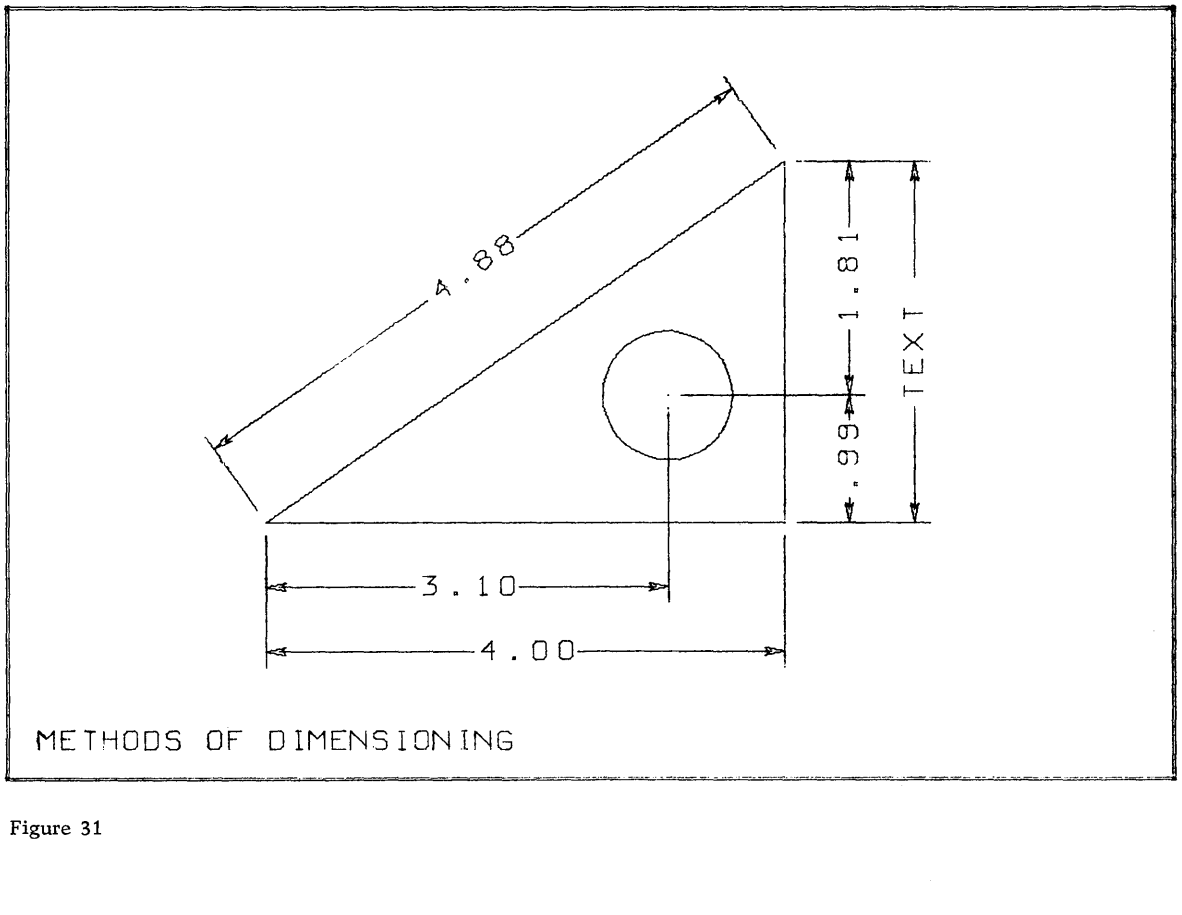


Figure 31

The DIMCR statement is used to dimension circles and arcs. This statement causes:

1. Arrows to be drawn to the edges of the circle. The orientation of the arrows is determined by the scalar value written in the DIMCR statement. The placement of the arrows is determined by the values in a previous DIMP statement.

2. Text to be drawn. If no text is given in the DIMCR statement, the value of the radius is drawn. If the value of the diameter is desired, or if other textual information is desired, a literal with the desired text must be placed in the DIMCR statement. The position of the text is determined by the appropriate value in a previous DIMP statement. The format of the displayed text is determined by the values in a previous MASK statement.

   Figure 32 contains two examples of the results of a DIMCR statement. The circle in the middle of the part was dimensioned by the statements:

   ```
   DIMP/ .5, .1, 1, 1
   MASK/ @ H D1 @
   DIMCR/ C2, 0, @BORE@ $$DIMENSION CIRCLE C2
   ```

   The arc at the lower left corner of the part was dimensioned by the statements:

   ```
   DIMP/ -1.3, .2, 2, 0
   MASK/ @ H D1 @
   DIMCR/ (CIRCLE/ A1), 45, @.25R@ $$DIMENSION ARC A1
   ```

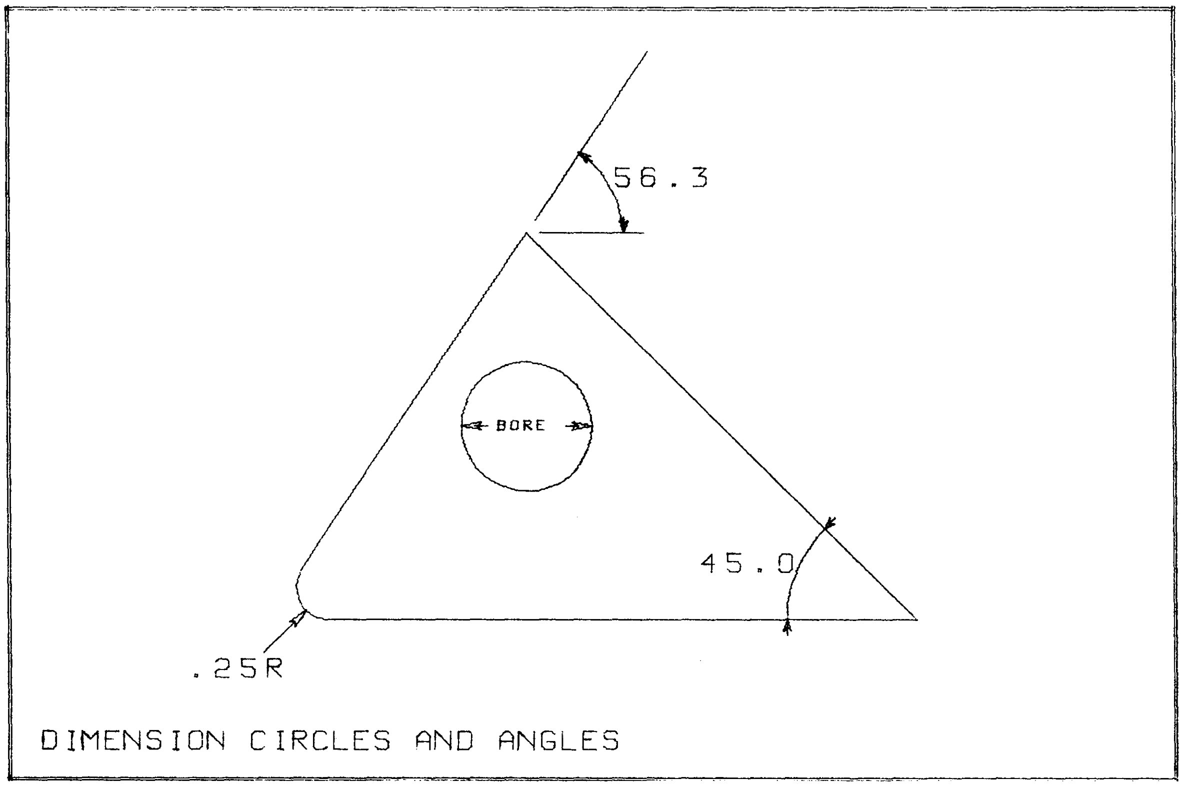


Figure 32

DIM/Angles — Dimension Angles

General Form:

DIM/ point, scalar, scalar, scalar, alpha

This special form of the DIM statement is used to dimension angles. The general form is equivalent to:

DIM/ point, paper radius, beginning angle, sweep angle, optional text

This form of the DIM statement causes:

1. An arc to be drawn with arrows at the end points of the arc. The center of the arc is the point given in the DIM statement. The radius of the arc, in paper inches, is the first scalar in the DIM statement. The end points of the arc are determined by the angles given in the DIM statement. Arrowheads are placed at the end points of the arc, pointing outward. If a negative sweep angle is given, the arrowheads point inward.

2. The value of the angle to be drawn. This value is the sweep angle given in the DIM statement. If a literal is written in the DIM statement, the text within the literal is drawn. The text is drawn always horizontal and is placed in the middle of the arc's sweep. The display of the text is determined by the values given in a previous MASK statement.

   Figure 32 contains two examples of the dimensioning of angles. The dimension at the lower right corner of the part was drawn by the statements:

```
MASK/ @ D1 @
DIM/ PR, 1, 180, -45    $$DIM THE ANGLE AT POINT PR
```

   The dimension at the top corner of the part measures the angle of the line labeled LX:

```
DIM/ PC, .75, 0, ATAND(LX)   $$DIM LINE LX AT POINT PC
```

2.05 CONTROL STATEMENTS

Control statements are used to direct the processing of 1620 Drafting System programs. There are three categories of control statements:

1. Systems Control Statements are used to perform miscellaneous functions during processing, such as typing a message to the computer operator or defining abbreviations for language words.

2. Looping Control Statements enable the programmer to execute language statements more than one time. For example, a grid consisting of a large number of lines can be drawn with just two line statements and appropriate Looping Control Statements.

3. Macro Control Statements are used to define and to execute macros. A macro is a general purpose program that is written, stored on the disk pack, and executed whenever needed.

2.05.01 SYSTEMS CONTROL STATEMENTS

PAUSE — Type a Message and Halt

General Form:

PAUSE/alpha

A PAUSE statement causes a temporary halt in processing during the drawing phase. The literal given in the statement is printed on the typewriter, and the computer is stopped; to resume processing, the operator presses the START button. For example, to notify the operator to change the size of the plotter's pen, this statement can be used:

```
PAUSE/@ ATTACH PEN NO. 7 @
```

INSERT — Output Plotter Commands

General Form:

INSERT/alpha

The INSERT statement allows a programmer to output actual plotter commands. The commands are written in the literal indicated in the statement. The INSERT statement allows the programmer to do such things as drawing special characters or operating special functions unique to his plotter.

For example, the command digit 9 causes the IBM 1627 Plotter to raise the pen; the command digit 0 causes the pen to be lowered. Therefore, the statement INSERT/ @ 9090909090 @ causes the pen to be raised and lowered five times. The commands for the 1627 Plotter are explained in the 1627 Plotter manual (A26-5710-0).

The drafting processor does not analyze the information in the literal. The programmer must ensure that the commands are valid and that the status of the pen is restored. Both the location and the up or down condition of the pen should be the same after the INSERT statement as before.

FINI — End of Program

General Form:

```
FINI/
```

FINI must always be the last statement of a program, and must be the only word appearing in the statement. This statement tells the Drafting System processor that there are no more statements in the program.

RESET — Change the Paper Origin

General Form:

RESET/<u>point</u>

Automatic setup for a new drawing can be programmed with the RESET statement. The point written in this statement is measured from the original paper origin. This statement causes the pen to be moved to the specified point. The point is then treated as a new paper origin.

Coding examples:

```
RESET/ 17,0

RESET/ 0, 11
```

The RESET statement is written normally as the last statement before the FINI statement, so that the plotter paper will be positioned properly for the next drawing.

DEFINE — Define a New Word

General Form:

label=DEFINE/<u>alpha</u>

The DEFINE statement enables a programmer to substitute his own word in place of a Drafting System word, phrase, or statement.

For example, to substitute XS for the language word XSMALL, write:

```
XS=DEFINE/ @XSMALL,@
```

After this DEFINE statement is written, the programmer can use XS anywhere that XSMALL would normally be used:

```
POINT/XS,INTOF,C1,C2
```

Writing a DEFINE statement does not destroy the original language word. Either XSMALL or XS can be used after the DEFINE statement.

The punctuation that follows a language word must appear in the DEFINE statement. In the DEFINE statement shown above, a comma is written after XSMALL. The punctuation occurring after the user's label (the comma after XS) is ignored; it may be any single punctuation character. The following examples are equivalent to the POINT statement shown above:

```
POINT/XS*INTOF,C1,C2

POINT/XS/INTOF,C1,C2
```

A phrase may be replaced in a similar manner; for instance:

```
H=DEFINE/ @LINE/DX,@
```

enables the programmer to use H in place of the phrase LINE/DX. The coder may now define a horizontal line five inches long by writing:

```
H/5
```

To replace an entire statement, follow the same procedure:

```
SIZE2=DEFINE/@ALPHAP/(.2,.2,.2,0),(0,-.2,0,0)@
```

After this DEFINE statement is written, the label SIZE 2 can be used in place of the entire ALPHAP statement.

Blank characters in language statements are usually ignored by the 1620 Drafting System. However, blanks may not appear within the literal in a DEFINE statement.

A45=DEFINE/@ATANGL, 45@ is valid.

A45=DEFINE/@AT ANGL, 45@ is invalid.

CALLAP — Call Application Program

General Form:

CALLAP/<u>scalar</u>

This statement makes it possible for a user-written program to be executed as part of the 1620 Drafting System. For example, a user might write a program that calculates the area of a surface. The Drafting System could then be used to describe a surface and to call the area program to perform the calculations. The information needed to write a program of this type is contained in the System Manual.

2.05.02 LOOPING CONTROL STATEMENTS

A program in the Drafting System consists of a series of statements. When these statements are processed by the system, each statement is executed in the same sequence as it appears in the written program. In some cases it is a significant advantage if the programmer can control and alter the sequence of execution.

The series of statements whose order of execution is controlled by the programmer is called a loop. A loop always contains one or more statements that have been assigned a statement number, which is always one of the digits 0 through 9. When a statement number is written, the digit is followed by a right parenthesis and the statement:

```
3)L1=LINE/DX,4
```

Within a loop, no two statements should be assigned the same number; however, the numbers in one loop may be repeated in another loop of the program.

LOOPST — Start of a Loop

General Form:

```
LOOPST/
```

The LOOPST statement signifies the start of a loop. All looping statements must be preceded by a LOOPST statement.

LOOPND — End of a Loop

General Form:

```
LOOPND/
```

A LOOPND statement must be given at the end of a series of statements constituting a loop.

JUMPTO — Go to a Statement

General Form:

```
JUMPTO/statement number
```

The JUMPTO statement is used to indicate the execution of a statement other than the next one in a program. The referenced statement may occur in the program before or after the JUMPTO statement. The statements in the following example are executed in the sequence: 1,2,5,6,3,4,7.

```
1) LINE/...
2) JUMPTO/5
3) LINE/...
4) JUMPTO/7
5) LINE/..
6) JUMPTO/3
7) LINE/...
```

IF — The IF Statement

General Form:

IF (scalar) statement number, statement number, statement number

An IF statement makes possible a choice for the next statement to be executed. The statement to be executed next is determined by the value of the scalar. If the value is negative, the first statement number is chosen, and the corresponding statement is executed after the IF statement. If the value is zero, the second statement number is chosen. The third statement number is selected for a positive value.

The statements in the following example are executed in the sequence: 1,2,3,4,2,3,4,5,6,2,3,4,7.

```
1) A=8
2) A=A + 1
3) LINE/...
4) IF (A-10)2,5,7
5) LINE/...
6) JUMPTO/2
7) LINE/...
```

2.05.03 MACRO CONTROL STATEMENTS

Macros greatly increase the power of the drafting language by permitting a programmer to obtain a large amount of drafting with a small amount of writing. Macros are useful especially in drawing items that occur frequently — rectangles, bolt hole circles, or title blocks on a drawing. Even more significant are certain commonly drawn subparts of a drawing, such as the fully dimensioned side view of a cylinder, or a complete drawing of a standard gear.

Each user of the drafting language will have a different set of frequently used drafting pictures. The user should identify these items, write out macros for them, and include the macros in the drafting system.

To develop macros, follow this procedure:

1. Identify a candidate for a macro. Any feature of a drawing that is being coded repeatedly in many drawings is a candidate. Rectangles are common examples.

2. Determine the variables of the macro. Analyze those items of the picture that vary from one drawing to another. The length and the width of a rectangle are examples of the variables needed for the macro of a rectangle. These variables are called parameters of the macro.

3. Program the macro. A macro is programmed in almost the same manner as a normal drawing. The significant difference is that the programmer codes the parameters of the macro as variables instead of as definite numbers. The length of a rectangle is coded as LEN, for example, in place of a constant number.

4. Store the macro in the macro library. A macro is stored as a part of the disk file library during a regular run of the drafting system. The only difference in a macro-definition run is that to produce a drawing, the program just written is stored instead of being executed.

5. Use the macro. Once the macro has been stored, it can be used by writing a language statement that calls the macro and assigns values to the parameters.

MACRO — Begin a Macro Definition

General Form:

```
Label=MACRO/scalar parameter, scalar parameter,...
```

The MACRO statement is used to define a macro, to assign a name to it, and to list the names of any scalar parameters for the macro. The label given to the MACRO statement is the name by which the programmer wishes to reference the macro. A maximum of ten scalar parameters can be listed in the MACRO statement.

Examples:

```
RECT=MACRO/ LEN, WID
SPUR=MACRO/N1, N2, N3
BORDER=MACRO/
```

TERMAC — Terminate a Macro

General Form:

```
TERMAC/
```

The TERMAC statement is written after the last statement in a macro program. This statement notifies the processor that the definition of a macro has terminated.

Example:

```
TERMAC/
```

DEFxx — Define the Parameters of a Macro

The scalar parameters of a macro are listed in the MACRO statement itself. Parameters of other types, however, are listed in special parameter-definition statements and follow the MACRO statement. The maximum number of parameters of all types (including scalars) that can be defined as part of a macro is ten. The following statements are used to identify various parameters:

```
DEFPT/label of point parameter,...
DEFCR/label of circle parameter,...
DEFLN/label of line parameter,...
DEFAR/label of arc parameter,...
DEFVU/label of view parameter,...
DEFSH/label of shape parameter,...
DEFLT/label of lateral parameter,...
```

Examples:

```
DEFPT/P1, P2, P3, P4
DEFCR/C55
DEFVU/BLOCK, SIDE
```

When these statements are used, they must follow the MACRO statement but precede all other accompanying statements.

CALL — Call a Macro for Execution

General Form:

```
CALL/macro name,,parameter name=value,,parameter name=value,,...
```

Once a macro has been defined and stored in the library, it can be used in any program by writing a CALL statement. The purpose of the CALL statement is to tell the processor which macro is being called and what values are to be given to the parameters of the macro.

The macro name must be the name used to define and store the macro.

A parameter name is the label assigned to the parameter in the macro definition. The name is followed by an equal sign and by the value that is to be assigned to the parameter. The value can be any drafting language expression, but no equal signs may appear in a CALL statement, except those used to set the parameter values. Note that double commas must separate each phrase in a CALL statement.

In each program the first CALL statement written for a stored macro must include values for all of the parameters of the macro. Any other CALL statement written for the same macro, and before a different macro is called, need include only those values for parameters that have changed. The order of the parameters in a CALL statement is unimportant.

Examples:

```
CALL/RECT,,LEN=2,,WID=1
CALL/RECT,,WID=1,,LEN=2
CALL/RECT,,LEN=3.7
CALL/RECT,,LEN=4 * SQRT (N5),,WID=N9
CALL/HOLE,,CTR=POINT/2,2,,ID=.3,,TEXT=@.3DIA@
```

The following statements define a macro for a rectangle.

```
RECT=MACRO/LEN,WID

LINE/DX,LEN

LINE/DY,WID

LINE/DX,-LEN

LINE/DY,-WID

TERMAC/
```

The first statement names the macro RECT and lists the two parameters of the macro. The horizontal length has been labeled LEN, and the vertical width called WID. The four line statements define a rectangle whose dimensions are the variable parameters LEN and WID. The TERMAC statement terminates the macro definition.

The macro can be used to draw rectangles as soon as it has been processed by the Drafting System and stored in the macro library. Macro definition programs are processed in the same manner as regular drawing programs. The macro shown above can be processed (and thereby stored) by adding a FINI statement after the TERMAC statement, and by then entering all of the statements into the computer. Any number of macros can be stored during the same computer run. Furthermore, a Drafting System program can contain several macro definitions as well as a regular drawing program. In such a case, all of the macro definition statements must precede the regular program statements.

To use a macro, write a CALL statement:

```
CALL/RECT,,LEN=2,,WID=1
```

This call of the macro RECT sets the length equal to two inches, and the width equal to one inch. Whenever a programmer wishes to draw a rectangle, he need only code a CALL statement. In effect, the ability to define rectangles in a single statement has been added to the Drafting System language.

The rectangle macro shown above has two scalar parameters — LEN and WID. The following example shows how a circle can be added as another parameter of the macro:

```
RECT=MACRO/LEN, WID

DEFCR/CM

LINE/DX, LEN

LINE/DY, WID

LINE/DX, -LEN

LINE/DY, -WID

CIRCLE/ CM

TERMAC/
```

To call for the rectangle and the circle, statements such as the following can be used:

```
CALL/RECT,,LEN=3,,WID=2,,CM=CIRCLE/P1,.25

CALL/RECT,,LEN=3,,WID=2,,CM=P1, .25

CALL/RECT,,LEN=3,,WID=2,,CM=CIRCLE/7,4,N1/2
```

In the preceding examples, the parameters are used to submit values to the macro. A parameter can also be used to return a value to the calling program. The following example shows how a macro can compute the area of an object and return the value of the area to the calling program through the parameter AREA.

```
RECT=MACRO/LEN, WID AREA

DEFCR/ CM

LINE/DX, LEN

LINE/ DY, WID

LINE/ DX, -LEN

LINE/ DY, -WID

CIRCLE/ CM

R=PARAM (3, CM) $$EXTRACT THE RADIUS OF THE CIRCLE

$$ COMPUTE THE AREA OF THE RECTANGLE MINUS THE AREA OF THE CIRCLE

AREA=LEN * WID -3. 14159 * R * R

TERMAC/
```

When this macro is called, the parameter AREA must be assigned a label:

```
CALL/RECT,,LEN=3,,WID=2,,CM=CIRCLE/7,4,.5,,AREA=AR
```

A subsequent statement can use the value of the area:

```
VOL=AR * 1.4
```

Any statement in the Drafting System language can appear in a macro, except a MACRO or a CALL statement. The names given to the parameters of a macro apply only to that macro and not to other macros or a calling program. For example, several macros that have a parameter labeled LEN can be added to the 1620 Drafting System.

2.06 EXAMPLES OF PROGRAMMING

The best way to code a drawing is as follows:

1. Choose a view and write a VIEW statement. A view in the 1620 Drafting System can be defined as a set of object lines and annotative information that are interdependent. Pick some point in the view as the part origin. The best point to choose is the one from which the dimensions of the part are expressed most easily.

2. Code the object lines by writing appropriate statements for all of the geometric elements. The object lines are those straight lines, circles, and arcs which describe the part view to be drawn.

3. Choose a scale factor and placement position on the paper for the view, and write a SCALE statement and an ORIGIN statement.

4. Indicate to the processor that drawing is to begin by writing a DRAW statement.

5. Annotate the view by writing note and dimension statements.

6. If a drawing has several views, choose the next view to be described, and go back to step 1. When the last view has been described, write a FINI statement.

2.06.01 THREE ADDITIONAL LANGUAGE WORDS

The 1620 Drafting System contains short forms for the words POINT, LINE, and CIRCLE. These additional words are PT, LN, and CR. The statement PT/1,1 is exactly equivalent to POINT/1,1. The short forms are provided because of the frequency of occurrence of point, line, and circle definitions. All other words in the language must be written exactly as presented in this manual.

2.06.02 SAMPLE DRAWINGS

Figure 33 is an example of a drawing with two views. The program statements used to produce the drawing are shown in Figure 34. The sequence in which the program statements were written follows the six steps presented above. Each view is described, drawn, and annotated.

An example of a grid is drawn in Figure 35. The program statements written to produce the drawing are shown in Figure 36. The program illustrates the basic use of looping. The eleven vertical lines were drawn by the first loop in the program. The second loop defines the eleven horizontal lines. The second loop draws two lines each time it is executed. In that manner the drawing time is shortened, because each line is drawn from the point nearest the end of the previous line.

Figure 37 shows a more complex example of the looping feature of the language. The language statements for the drawing are shown in Figure 38. The top portion of the drawing is done by drawing three lines each time a loop is executed. The lower portion of the drawing is symmetrical to the top half; therefore, a MIRY statement is used to draw the mirror image of the entire top portion of the drawing.

Figure 39 depicts the definition of a macro which can be used to draw slots. The statements written to define the macro are shown in Figure 40. The macro is designed so that the programmer who uses it need specify only the width of the slot and the centerline of the slot. The statements which define the macro construct the four points P1, P2, P3, and P4 in terms of the given line and width. A REFSYS statement is used in this definition of the macro so that the macro can be used easily in any orientation.

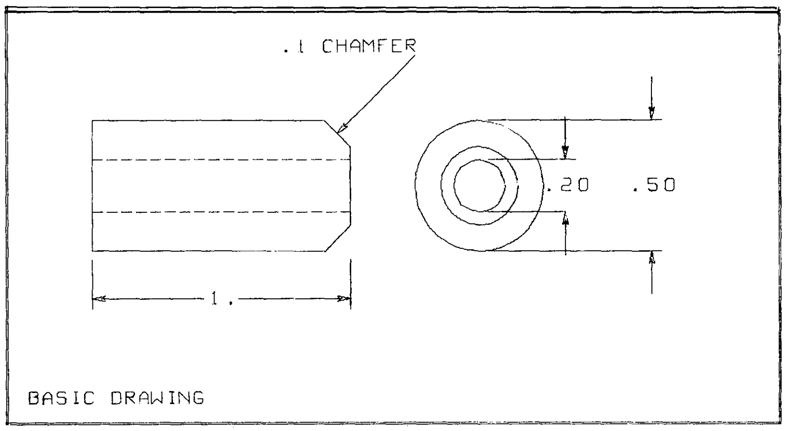


Figure 33

```
$$BASIC DRAWING
SCYL=VIEW/                          $$FRONT VIEW
LINE/DX,.9                          $$BOTTOM EDGE
LINE/DX,.1,DY,.1                    $$CHAMFER
LINE/DY,.3                          $$RIGHT EDGE
LINE/DX,-.1,DY,.1                   $$CHAMFER
LINE/DX,-.9                         $$TOP EDGE
LINE/DY,-.5                         $$LEFT EDGE
LB=DASHED,LN/0,.15,DX,1             $$LOWER DASHED LINE
DASHED,LN/YLARGE,PARLEL,LB,.2       $$UPPER DASHED LINE
END/SCYL
$$
SCALE/3
ORIGIN/1,2
$$
DRAW/SCYL
$$
DIMST/YSMALL,XCOMP,LB ,1            $$DIM PARALLEL TO BOTTOM
DIM/0,0,1,0                         $$DRAW ONE INCH DIMENSION
NOTER/.95,.45,1,1,'.1 CHAMFER'      $$DRAW NOTE
$$
ECYL=VIEW/                          $$SIDE VIEW
CIRCLE/0,0,.25                      $$OUTER CIRCLE
CIRCLE/0,0,.15                      $$MIDDLE CIRCLE
CIRCLE/0,0,.1                       $$CENTER CIRCLE
END/ECYL
$$
ORIGIN/5.5,2.75
$$
DRAW/ECYL
$$
LDUM=LINE/0,-.25,0,.25              $$DUMMY LINE FOR DIM
DIMST/YCOMP,LDUM                    $$DIM PARALLEL TO DUMMY LINE
DIMP/.5,.2,2,2                      $$REVERSE ARROWS
MASK/'H D2'                         $$TWO DECIMAL PLACES
DIM/0,-.1,0,.1                      $$DIM INNER DIAMETER
DIM/LDUM                            $$DIM OUTER DIAMETER
FINI/
```

Figure 34

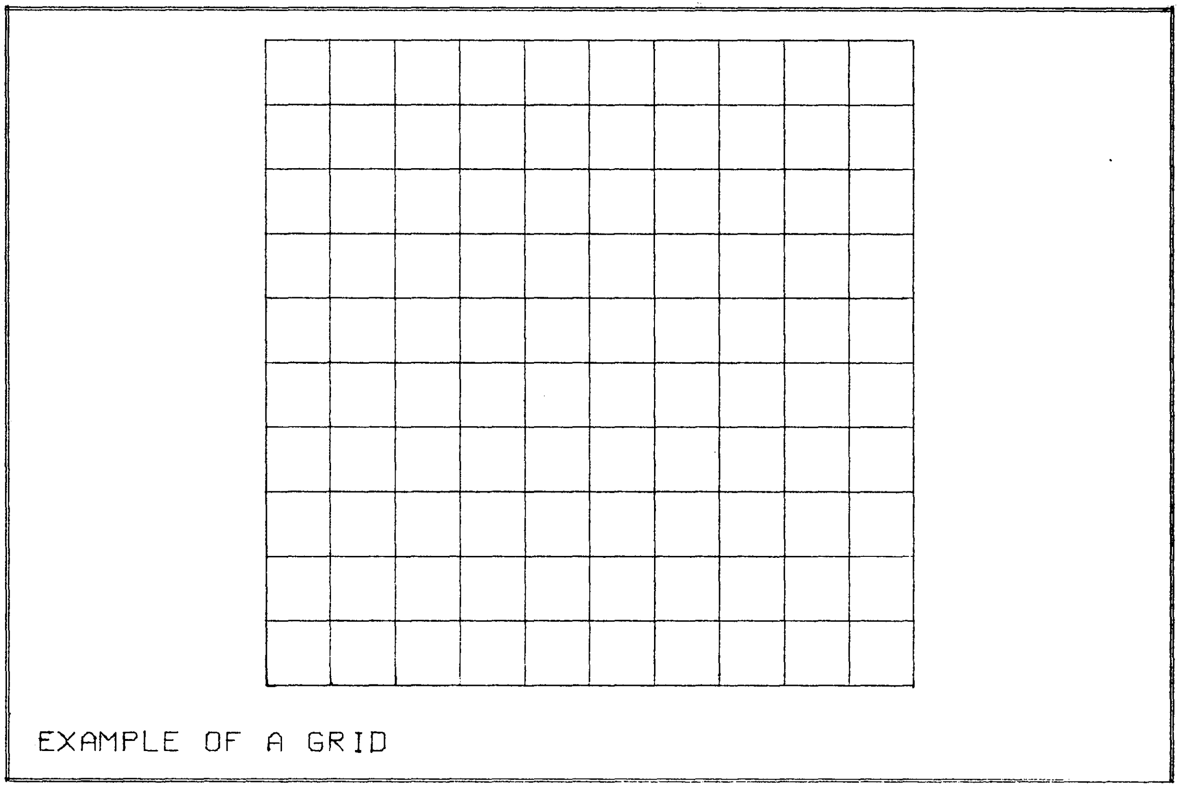


Figure 35

```
$$EXAMPLE OF A GRID
GRID=VIEW/                        $$BEGIN A VIEW
X=0                               $$INITIAL X-VALUE
LOOPST/                           $$START LOOP
1)LINE/X,0,DY,10                  $$VERTICAL LINE
X=X+1                             $$INCREASE X BY 1
IF(X-10)1,1,2                     $$IS X GREATER THAN 10
2)LOOPND/                         $$YES,END OF LOOP
LINE/0,0,DX,10                    $$BOTTOM LINE
$$
Y=1                               $$INITIAL Y-VALUE
LOOPST/                           $$START SECOND LOOP
1)LINE/0,Y,DX,10                  $$HORIZONTAL LINE
LINE/10,Y+1,DX,-10                $$REVERSE DIRECTION LINE
Y=Y+2                             $$INCREASE Y BY 2
IF(Y-9)1,1,2                      $$IS Y GREATER THAN 9
2)LOOPND/                         $$YES,END OF LOOP
END/GRID                          $$END THE VIEW
$$
SCALE/.5
ORIGIN/2,.75
$$
DRAW/GRID                         $$DRAW THE VIEW
FINI/
```

Figure 36

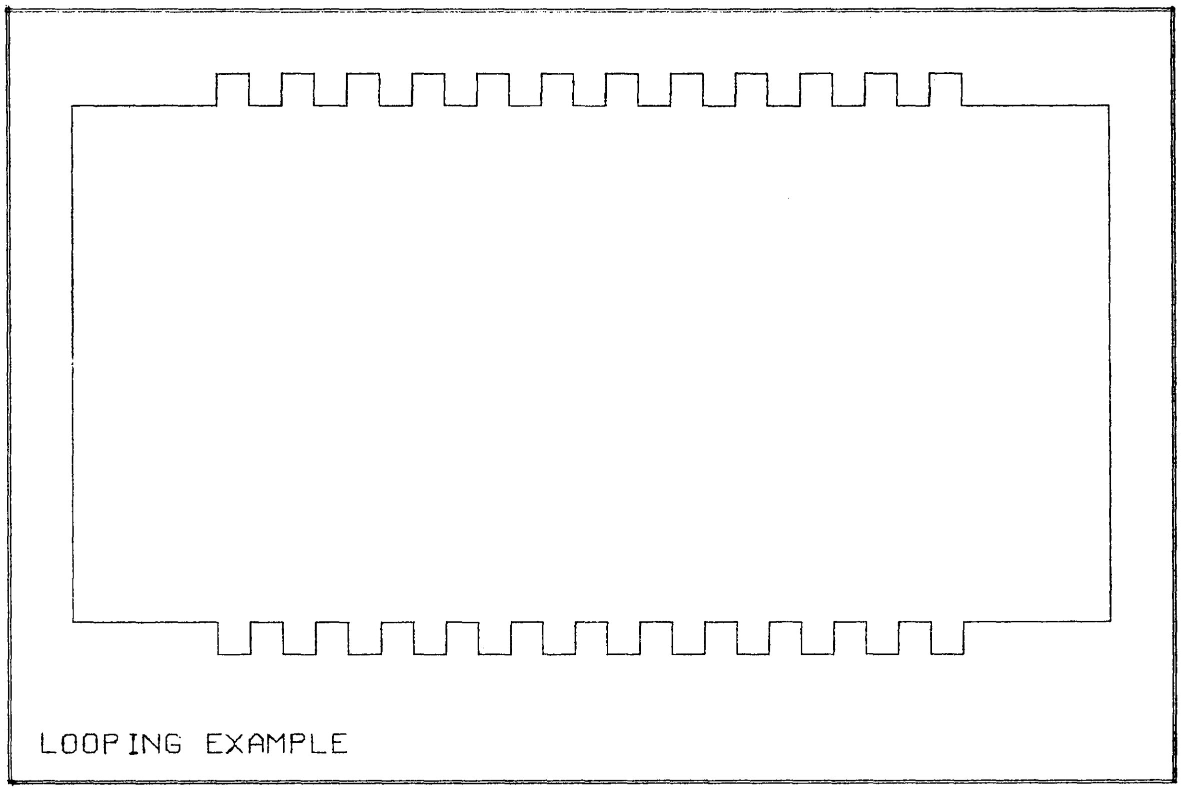


Figure 37

```
$$LOOPING EXAMPLE
TOP=VIEW/                         $$DEFINE THE TOP VIEW
LINE/DY,2                         $$LEFT EDGE
LINE/DX,1+1/8                     $$TOP LINE
CTR=12                            $$SET A COUNTER TO 12
LOOPST/                           $$BEGIN A LOOP
1)LINE/DY,.25                     $$VERTICAL LINE
LINE/DX,.25                       $$HORIZONTAL LINE
LINE/DY,-.25                      $$VERTICAL LINE DOWN
LINE/DX,.25                       $$HORIZONTAL LINE
CTR=CTR-1                         $$DECREASE COUNTER
IF(CTR)2,2,1                      $$REPEAT 12 TIMES
2)LOOPND/                         $$END OF LOOP
LINE/PPP,8,2                      $$TOP LINE
LINE/PPP,8,0                      $$RIGHT EDGE
END/TOP                           $$END OF TOP VIEW
ORIGIN/.5,3.25
DRAW/TOP                          $$DRAW THE VIEW
DRAW/MIRY(TOP)                    $$DRAW THE MIRROR IMAGE
FINI/
```

Figure 38

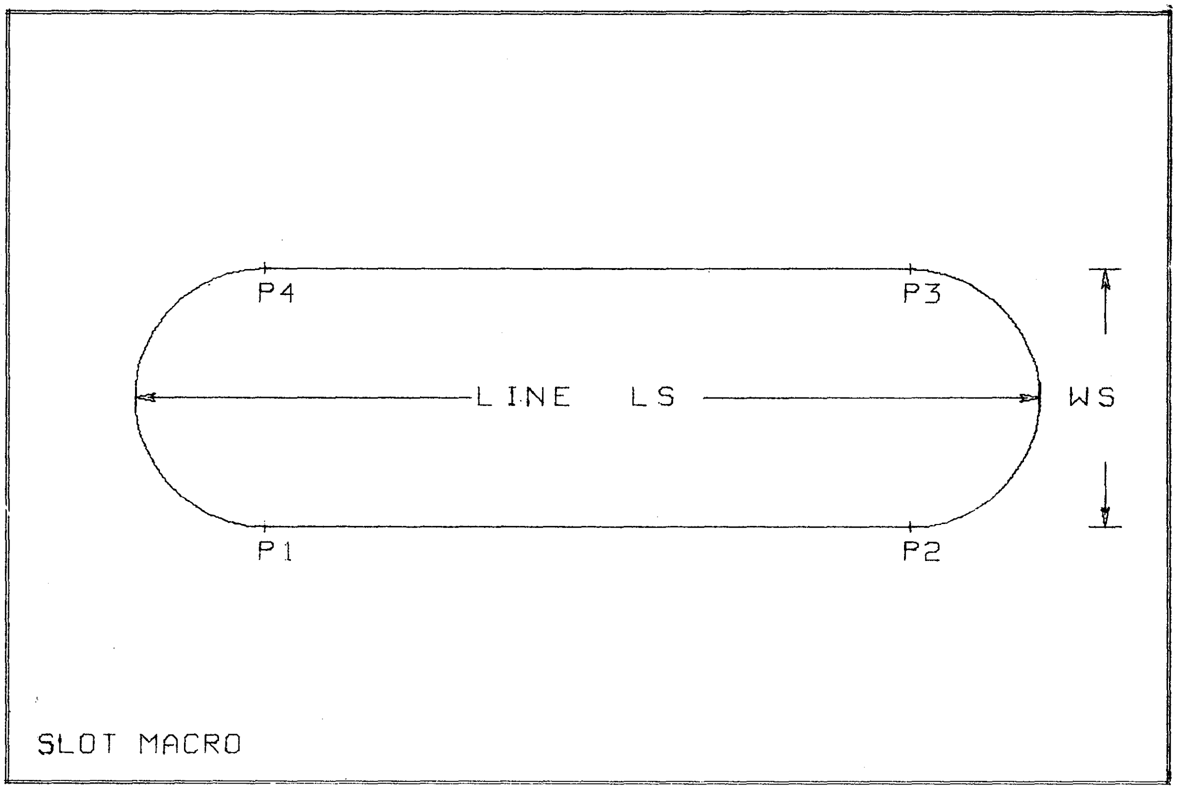


Figure 39

```
$$SLOT MACRO
SLOT=MACRO/WS                    $$MACRO NAME IS SLOT
$$WS IS A SCALAR PARAMETER
DEFLN/LS                         $$LS IS A LINE PARAMETER
WS2=WS/2                         $$HALF THE WIDTH
REFSYS/ANGOF(LS)                 $$LOCAL ORIGIN ALONG THE LINE LS
P1=PT/WS2,-WS2,,P2=PT/DIST(LS)-WS2,-WS2
P3=PT/MIRY(P2),,P4=PT/MIRY(P1)
LINE/P1,P2                       $$BOTTOM LINE
ARC/P2,P3,XSMALL,RADIUS,WS2      $$RIGHT ARC
LINE/P3,P4                       $$TOP LINE
ARC/P4,P1,RADIUS,WS2             $$LEFT ARC
REFSYS/NOMORE                    $$RELEASE REFSYS MODE
TERMAC/
```

Figure 40

Some examples of the macro defined in Figure 40 are illustrated in Figure 41. The statements written to call the macro are shown in Figure 42. A single CALL statement was written for each appearance of the slot.

2.07 DIAGNOSTICS

Each phase of the processor of the 1620 Drafting System has its own error detection routines. The general philosophy for error routines is that they should type out an error message indicating each error condition and then continue as much processing as possible.

2.07.01 COMPILER PHASE

Whenever the compiler detects a source statement error, an error message is typed out, and processing is resumed. This procedure makes it possible for the compiler to check the remaining source statements for further errors. If any source statement contains an error, all processing after the compiler phase ceases, and the 1620 Monitor is called.

The *ERROR STOP control card (explained in the Operator's Manual) can be used so that whenever an error occurs during compilation, the statement in error can be corrected from the typewriter of the computer. With this option, if all errors have been corrected, normal operations can continue after the compilation phase.

If any of the tables or disk areas of the compiler are exceeded during compilation, the compiler types an error message and immediately calls the next phase of the 1620 Drafting System.

Some errors terminate the job immediately and cause the 1620 Monitor to be called. A source statement that attempts to call a nonexistent macro will cause such an error.

Error message 190 — too many numbers in a loop — results from a special condition. The compiler keeps a table of numbers. A table can contain 50 numbers. When a table is filled, it is written out onto the disk file, and the compiler begins to construct another table. Within a loop, however, if a table of numbers is exceeded, the compiler cannot function properly, and an error message is given. It is possible to avoid the error condition by writing a special form of the LOOPST statement:

LOOPST/N

The addition of the letter N to the LOOPST statement should be done only if error 190 has occurred.

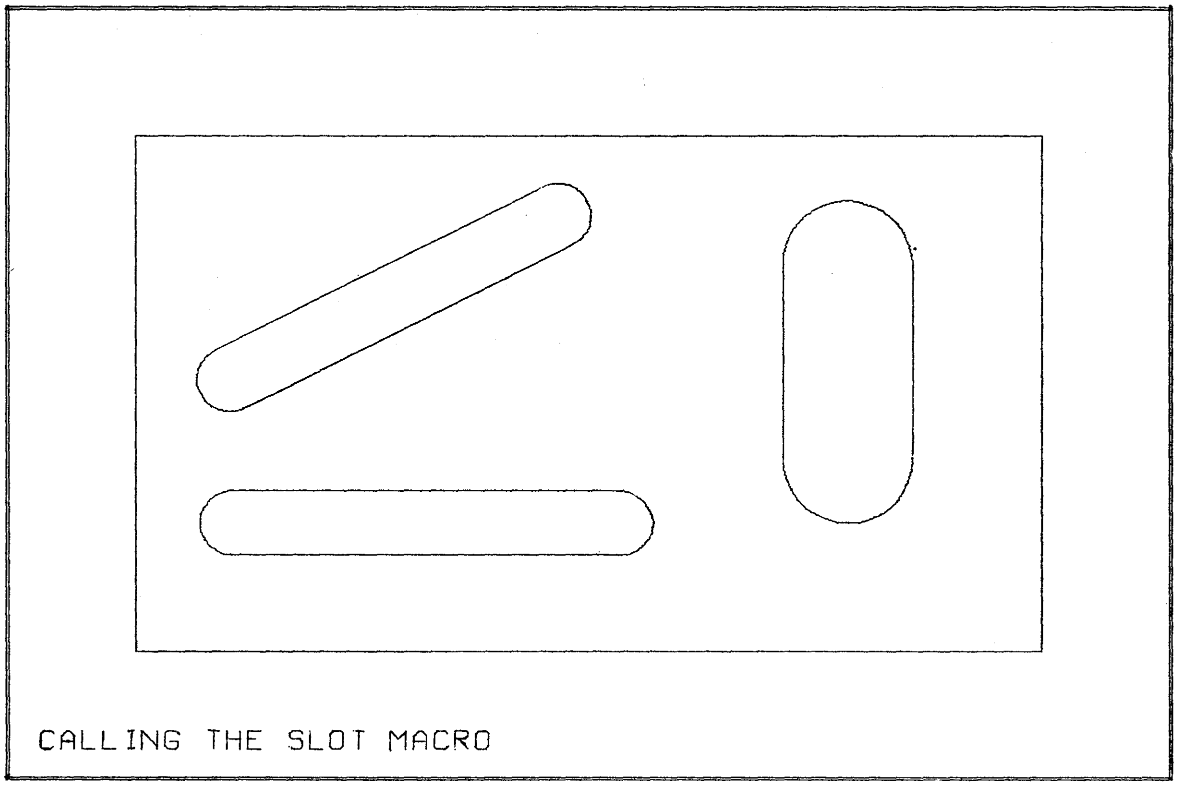


Figure 41

```
$$CALLING THE SLOT MACRO
PIX=VIEW/                                 $$BEGIN A VIEW
LINE/DX,7                                 $$BOTTOM EDGE OF PART
LINE/DY,4                                 $$RIGHT EDGE
LINE/DX,-7                                $$TOP EDGE
LINE/DY,-4                                $$LEFT EDGE
$$
CALL/SLOT,,LS=(.5,1,4,1),,WS=.5           $$HORIZONTAL SLOT
CALL/SLOT,,LS=(.5,2,3.5,3.5 )             $$OBLIQUE SLOT
CALL/SLOT,,LS=(5.5,1,5.5,3.5),,WS=1       $$VERTICAL SLOT
END/PIX                                   $$END THE VIEW
$$
ORIGIN/1,1
$$
DRAW/PIX                                  $$DRAW THE VIEW
FINI/
```

Figure 42

2.07.02 PART PROCESSOR PHASE

The part processor writes out two sets of disk records — a part model file and a drawing command file. If, during processing, either of these file areas is exceeded, the phase is terminated immediately, and the drawing processor is called to execute as much of the program as the file area can hold.

Whenever a geometric construction cannot be made from the data of the input statement, a standard error form is substituted, an error message is typed out, and processing is resumed. These standard forms are equivalent to the language statements:

```
POINT/2,2

CIRCLE/2,2,1

LINE/2,2,1,1

ARC/2,2,1,1,30
```

2.07.03 DRAWING PROCESSOR PHASE

Whenever possible, the drawing phase bypasses an error and continues processing. A serious error causes immediate termination of the job, and the 1620 Monitor is called. If the user desires to attempt processing after a serious error, he can reexecute the job with switch 1 on, in which case the drawing processor tries to continue processing beyond the error condition.

2.07.04 MAXIMUM TABLE SIZES

Total number of labels, including those used for scalars	100
Labels of scalars	25
Literals	300
Total number of macros	100
Number of parameters in a macro	10

2.08 DICTIONARY

Language Word	Section(s)	Meaning
ABS	2.03.02	Arithmetic function for the absolute value of a scalar.
ALOG	2.03.02	Arithmetic function for the natural log of a scalar.
ALPHAP	2.04.03	Modal major word which sets parameters for lettering.
ANGOF	2.02.07 2.04.01	Nonmajor word used to set a reference line.
ARC	2.02.05	Major word to define an arc.
ATAND	2.02.06 2.03.02	Function word to determine the arc tangent in degrees. Its argument can be the slope of a line or a line definition.
ATANGL	2.02.07 2.04.01	Nonmajor word that indicates the next word in the statement is an angle.
CALL	2.05.03	Major word used to call for the execution of a macro.
CALLAP	2.05.01	Major word used to call for the execution of an application program written by the user.
CENTER	2.02.02	Word used in point definitions to select the center of a circle or an arc.
CIRCLE	2.02.04	Major word to define a circle.
CLW	2.02.05	Used in arc definitions to indicate that the desired arc sweeps clockwise from the initial point to the ending point.
CONSTR	2.02.01	Line class word to indicate a construction line.
COSD	2.03.02	Arithmetic function for the cosine of an angle given in degrees.
CR	2.06.01	Short form of the word CIRCLE.
CTRLN	2.02.01	Line class word to indicate a centerline.
CUTPL	2.02.01	Line class word to indicate a cutting-plane line.
DASHED	2.02.01	Line class word to indicate a dashed line.

DEFAR	2.05.03	Major word that lists the parameters of a macro that are arcs.
DEFCR	2.05.03	Major word that lists the parameters of a macro that are circles.
DEFINE	2.05.01	Major word that is used to substitute a user's label for a language word or phrase.
DEFLN	2.05.03	Major word that defines the parameters of a macro that are lines.
DEFPT	2.05.03	Major word that defines the parameters of a macro that are points.
DEFSH	2.05.03	Major word that defines the parameters of a macro that are shapes.
DEFVU	2.05.03	Major word that defines the parameters of a macro that are views.
DIM	2.04.05	Major word used to draw dimensions.
DIMCR	2.04.05	Major word used to dimension a circle.
DIMEE	2.04.05	Major word (equivalent to DIM) used to draw dimension lines with two extension lines.
DIMEN	2.04.05	Major word used to draw dimension lines with a single extension line at the "from" point.
DIMNE	2.04.05	Major word used to draw dimension lines with a single extension line at the "to" point.
DIMNN	2.04.05	Major word used to draw dimension lines with no extension lines.
DIMP	2.04.05	Modal word which specifies the location and size of the dimensioning text, and the orientation of the arrows.
DIMST	2.04.05	Modal word which specifies the type of dimensioning, the starting line, the indexing distance, and the direction of automatic indexing for subsequent DIM statements.
DIST	2.02.06	Geometric function that calculates the distance between two points.
DOTTED	2.02.01	Line class word used to indicate a dotted line.

DRAW	2.04.02	Major word that causes object lines to be drawn.
DX	2.02.03	Nonmajor word that indicates the X-component distance for a line.
DXOF	2.02.06	Geometric function that calculates the difference between the X-values of the end points of a line.
DY	2.02.03	Nonmajor word that indicates the Y-component distance for a line.
DYOF	2.02.06	Geometric function that calculates the difference between the Y-values of the end points of a line.
END	2.02.01	Major word that indicates the end of a geometric group.
EXP	2.03.02	Arithmetic function that calculates the exponential function value.
EXTEN	2.02.01	Line class word used to indicate an extension line.
FINI	2.05.01	Systems control word that indicates the end of all source statements in a program.
HATCH	2.04.02	Major word used to draw crosshatch lines.
HATCHP	2.04.02	Modal word that specifies the angle and spacing for crosshatch lines.
IF	2.05.02	Language word used to control the sequence of instruction execution, depending upon the value of an arithmetic expression.
INDEX	2.04.05	Major word used to override the automatic indexing feature in dimensioning.
INSERT	2.05.01	Systems control word used to output plotter commands.
INTOF	2.02.03	Nonmajor word used to indicate "intersection of"; the word is followed by the two geometric elements whose intersection is desired.
JUMPTO	2.05.02	Major word that alters the sequence of execution of language statements.
LEFT	2.02.03	Nonmajor word that specifies the left tangency condition for a line and a circle.

LENGTH	2.02.03	Nonmajor word indicating that the next number in the statement is the desired length of a line.
LINE	2.02.03	Major word used to define a line.
LN	2.06.01	Short form of the word LINE.
LOOPND	2.05.02	Control word that signifies the end of a loop.
LOOPST	2.05.02	Control word that signifies the beginning of a loop.
MACRO	2.05.03	Major word indicating that a macro is to be defined.
MASK	2.04.05	Modal word that controls the way in which the text for dimensioning is displayed.
MEDIUM	2.02.01	Line class word used to indicate a solid line of medium weight.
MIRX	2.02.06 2.04.02	Geometric function that defines the mirror image of a point, reflected about the vertical axis.
MIRXY	2.02.06 2.04.02	Geometric function that defines the mirror image of a point, reflected about both axes.
MIRY	2.02.06 2.04.02	Geometric function that defines the mirror image of a point, reflected about the horizontal axis.
NOMORE	2.02.07 2.04.01	Nonmajor word used to indicate the end of a mode.
NOTE	2.04.04	A major word used to draw notes.
NOTER	2.04.04	A major word used to draw notes that have an arrow extending from the right side of the text to the part.
ORIGIN	2.04.01	A modal word that locates a view on the drawing paper.
PARAM	2.02.06	Geometric function used to extract a value from the standard form of a point, line, circle, or an arc.
PARLEL	2.02.03	Nonmajor word used to construct parallel lines.
PAUSE	2.05.01	Systems control word that causes a message to be typed and the computer to be stopped temporarily.
PERPTO	2.02.03	Nonmajor word used to construct perpendicular lines.
POINT	2.02.02	Major word used to define a point.

PPP	2.02.01	The label of a point defined internally by the 1620 Drafting System as the Present Part Position.
PT	2.06.01	Short form of the word POINT.
RADIUS	2.02.05	Nonmajor word used in ARC definitions to indicate the radius of an arc.
REFSYS	2.02.07	Major word used to specify a local coordinate system for part coordinates.
RESET	2.05.01	Systems control word that resets the plotter for another drawing.
RIGHT	2.02.03	Nonmajor word used to indicate the right tangency condition of a line and a circle.
SCALE	2.04.01	Modal word which specifies the scale of a drawing.
SHAPE	2.02.01	Major word used to begin a grouping of geometric definitions.
SIND	2.03.02	Arithmetic function which calculates the sine of an angle given in degrees.
SQRT	2.03.02	Arithmetic function which calculates the square root of a number.
TANTO	2.02.03 2.02.04	Nonmajor word indicating that the construction of a tangency is desired.
TERMAC	2.05.03	Major word which terminates the definition of a macro.
THICK	2.02.01	Line class word that indicates a solid line of thick weight.
THIN	2.02.01	Line class word that indicates a solid line of thin weight.
TILLX	2.02.03	Nonmajor word that specifies the terminating X-value of a line.
TILLY	2.02.03	Nonmajor word that specifies the terminating Y-value of a line.
TITLE	2.04.04	Major word used to draw general notes.
TRUE	2.04.05	Nonmajor word used to specify the calculation of true dimensions.

VIEW	2.02.01	Major word used to begin a grouping of geometric definitions.
XCOMP	2.04.05	Nonmajor word used to specify the calculation of the X-component of a dimension.
XLARGE	2.02.03 2.02.04 2.04.05	Nonmajor word used to select one of two possible geometric constructions — XLARGE selects the condition with the largest X-value.
XSMALL	2.02.03 2.02.04 2.04.05	Nonmajor word used to select one of two possible geometric constructions — XSMALL selects the condition with the smallest X-value.
YCOMP	2.04.05	Nonmajor word used to specify the calculation of the Y-component of a dimension.
YLARGE	2.02.03 2.02.04 2.04.05	Nonmajor word used to select one of two possible geometric constructions — YLARGE selects the condition with the largest Y-value.
YSMALL	2.02.03 2.02.04 2.04.05	Nonmajor word used to select one of two possible geometric constructions — YSMALL selects the condition with the smallest Y-value.

H20-0183-0

Printed in U.S.A. H20-0183-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601