WANG

2200 Plotter Utilities Manual

2200 Plotter Utilities Manual

WANG LABORATORIES, INC.

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

HOW TO USE THIS MANUAL

The 2200 Plotter Utilities Manual provides operating instructions and program descriptions for each of the Plotter Utility subroutines. The manual does not, however, undertake to explain or discuss the operation of individual plotters; general information on plotter operation and control can be found in the appropriate plotter reference manual. It is recommended that the programmer be familiar with the operation of his plotter before attempting to use the Plotter Utilities.

The Plotter Utility Package currently is available in two versions, a disk version and a tape version*. The disk version may be obtained on a Model 2270-type diskette or a minidiskette. Package numbers for all three media are listed below. Be sure the version you have received is the proper one for your installation.

Tape cassette	- 195-0021-1
Model 2270-type diskette	- 195-0021-3
Minidiskette	- 195-0021-8

A particularly noteworthy feature of the Plotter Utility Package is the versatile alphanumeric labelling capability provided. All figures in the body of this manual were plotted and labelled with the Plotter Utilities on a Model 2232B Digital Flatbed Plotter.

* The tape version of the Plotter Utilities does not contain the Enhanced Character Set or the provisions described in Chapter 11.

TABLE OF CONTENTS

List of Figures

PAGE

List of Figures (Cont.)

List of Tables

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION TO THE PLOTTER UTILITIES

The Wang Plotter Utilities Package consists of a group of utility subroutines designed to perform a variety of commonly used plotting operations and to simplify general plotter control for the application programmer. The utility routines are written not as stand-alone programs (although they can be accessed directly from the keyboard via the Special Function Keys), but as subroutines which can be called from a user-supplied application program.

The subroutines perform a variety of functions. One routine permits the user to define a portion of the plotting surface within which all plotting will take place (the "active plotting area"). Other routines perform such commonly used operations as plotting a straight line, plotting a circle, plotting a coordinate grid, and plotting an alphanumeric character string. A generalized routine called the "Plot Instruction Emulator" simulates nine primitive plotting instructions; it is a powerful and versatile tool for writing customized plotter routines.

Memory Requirements and Minimum System Configuration

The plotter utilities require the following minimum system configuration:

Memory	- 12K minimum; 16K recommended.
CPU Options	- Sort Statements, Plot Statements, and Disk Statements
Peripherals	- Disk drive and/or tape cassette drive, any plotter. (With the addition of a special interface, a Tektronix graphic display also is supported.)

Total memory requirement for the plotter utilities, including all utility programs and all variables, is approximately 6K. Since the programmer has the option to load any combination of selected utilities at any time, and since occasions when all utilities must be loaded simultaneously are rare, the 6K figure is somewhat larger than the actual requirement for most applications. The procedure for determining the amount of memory required for a particular combination of utilities is described in Chapter 2.

Wang manufactures several different types of plotters, each with its own unique characteristics and idiosyncrasies. The plotter utilities are compatible with all Wang plotters. The utilities have been designed to obviate, to the extent possible, all dependence on individual plotter hardware. Only two utility routines, the Plotter Control and the Clear Surface/Pen Select routines, directly access the plotter, and are directly hardware dependent. The remaining routines are totally independent of the hardware and in general function identically for all plotters as well as the graphic display. Several versions of the Plotter Control Routine are included in the utility package: one version supports the Model 2202 plotting output writer, a second version supports the Model 2212 and Model 2232B digital flatbed plotters, a third version supports the Model 2272-2 drum plotter, and the Model 2282 Graphic CRT, a fourth version supports the Tektronix graphic display terminal, and a fifth version supports the Model 2281P.

Summaries of the Plotter Utility Routines

Each of the plotter utility programs is briefly summarized below:

1. SET PLOTTER BOUNDARIES (DEFFN'19)

   The Set Plotter Boundaries routine is used to define the boundaries of the "active plotting area." The "active plotting area" is that portion of the physical plotting surface available for plotting. Once defined, this area is recognized by all subsequent plotter utilities, and no plotting is permitted beyond its boundaries. (The boundaries can, however, be altered at any time under program control.)

2. LOAD CHARACTER GENERATION ARRAY (DEFFN'22)

   This routine loads a previously created character generation array from disk into memory. The character generation array is created initially by the START module during system startup and is stored out on disk or tape at that time. Two character generation arrays are available with disk versions the Plotter Utilities System. One array contains coded plotting sequences for 63 English alphanumeric characters and symbols. The other array contains coded plotting sequences for Greek characters and special electronic symbols. Each array must be loaded into memory prior to plotting a character string.

   The tape version of the Plotter Utilities System contains only the array for the 63 English alphanumeric characters and symbols.

3. PLOT CHARACTER STRING (STRAIGHT LINE) (DEFFN'20)

   The Plot Character String (Straight Line) routine plots a string of alphanumeric characters on a straight line called the "character base line." The characters are defined in the character generation array, which must be resident in memory to run this routine. The character size is specified by the user, as are the character slant and rotation. The base line coordinates also are specified by the programmer.

4. PLOT CHARACTER STRING (ON A CIRCLE) (DEFFN'21)

   This routine plots a string of alphanumeric characters on the circumference of a circle. The center point and radius of the circle are determined by the user, as are the character size and slant. The character string is plotted relative to a reference point on the circle.

5. PLOT LINE BETWEEN TWO POINTS (DEFFN'25)

   The Plot Line routine plots a straight line between two points whose coordinates are specified by the user. The line may be solid, dashed, dotted, or dashed/dotted.

6. PLOT COORDINATE GRID (DEFFN'26)

   The Plot Coordinate Grid routine plots a coordinate grid of horizontal and vertical grid lines in the active plotting area. The origin point of the grid, along with the increments between successive horizontal grid lines and between successive vertical grid lines, are specified by the programmer.

7. PLOT CIRCLE (DEFFN'27)

   The Plot Circle routine plots a circle whose center point and radius length are defined by the user. The size of the straight-line segments used in approximating the circle also must be specified by the user.

8. PLOT BORDER AROUND ACTIVE PLOTTING AREA (DEFFN'28)

   This routine plots a border around the active plotting area defined in the Set Plotter Boundaries routine. The border may be plotted as a solid, dashed, dotted, or dashed/dotted line.

9. CLEAR SURFACE/PEN SELECT (DEFFN'24)

   When used with the Model 2282 Graphic CRT, this routine enables the programmer to automatically clear the screen and select PLOT or Erase modes of operation. On the Model 2272-2 plotter this routine automatically selects the desired pen. On all plotters this routine prompts the operator when a new plotting surface is needed and on all pen plotters it alerts the operator that a different pen must be mounted.

10. GIN Mode Routine (DEFFN'23)

   This routine enables the 2200 to receive graphic input in the form of screen coordinates from certain models of the Tektronix graphic display terminal. (The GIN Mode Routine is used only with a Tektronix graphic display, and is covered in Appendix G.)

11. PLOT INSTRUCTION EMULATOR (DEFFN'29)

This utility is a generalized routine which simulates nine basic plot instructions. The user is required to specify only three parameters: an X and a Y value, and an instruction code. The X and Y values may be interpreted as X and Y coordinates, or as X and Y deltas, depending upon the instruction code specified. The nine available instructions are listed below:

1. Send Plotter Home
2. Move Delta X, Delta Y (Tacit Move)
3. Move to X,Y (Tacit Move)
4. Move Delta X, Delta Y (Actual Move)
5. Move to X,Y (Actual Move)
6. Move Delta X, Delta Y Plot Point
7. Move to X,Y (Plot Point)
8. Plot Line Delta X, Delta Y
9. Plot Line to X,Y

12. PLOTTER CONTROL ROUTINE (DEFFN'30)

The Plotter Control Routine is the only routine which addresses the plotter directly, and is the single device-dependent routine. It is called by the Plot Instruction Emulator to execute the plotter operations simulated by the instructions in the Emulator. Five versions of the Plotter Control Routine are provided: one version controls the Model 2212 and Model 2232B plotters, a second version controls the Model 2202 plotter, a third version controls the Model 2272-2 drum plotter and the Model 2282 Graphic CRT. A fourth version controls the Tektronix graphic display and a fifth version controls the Model 2281P. The Plotter Control Routine is not designed to be directly accessed by the user.

1.2 SYSTEM PHILOSOPHY AND DESIGN

The plotter utility package is intended to provide a powerful and versatile plotter control capability which also simplifies plotter control for the application programmer. A major design consideration was the isolation of the plotting routines from the idiosyncrasies of individual plotters. This intent to "universalize" the routines with respect to different plotters gave rise to a hierarchical structure in which routines at each level call routines on the next lower level to control the plotter, and only the Plotter Control Routine itself, on the lowest level, directly addresses the plotter. Figure 1-1 illustrates the hierarchy, showing four levels of control.

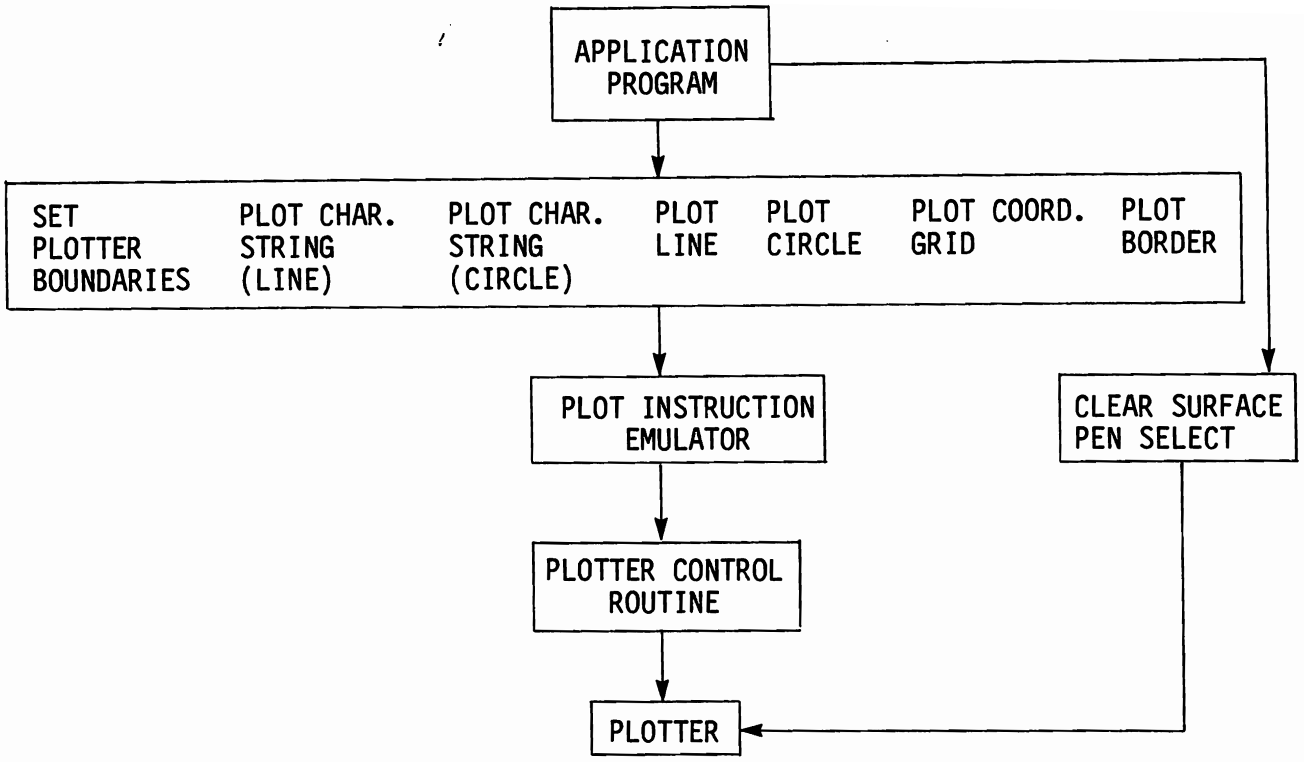


Figure 1-1. The Hierarchy of Plotter Control

As the figure illustrates, each of the special utilities calls the Plot Instruction Emulator to execute a specific plot. The Plot Instruction Emulator, in turn, calls the Plotter Control Routine, which translates each plot instruction into a sequence of PLOT statements and directly controls plotter movement. Because all direct plotter control originates in this routine, the Plotter Control Routine is the only device-dependent module in the utility package. All significant differences between the various plotters are compensated for in the Plotter Control Routine.

If the application programmer requires a plotter routine not provided by any of the special plot routines, he can bypass those routines and call the Plot Instruction Emulator directly from his mainline program. The nine plotter instructions simulated by the Plot Instruction Emulator provide a powerful tool for controlling plotter operation, with minimal "housekeeping" required by the application program.

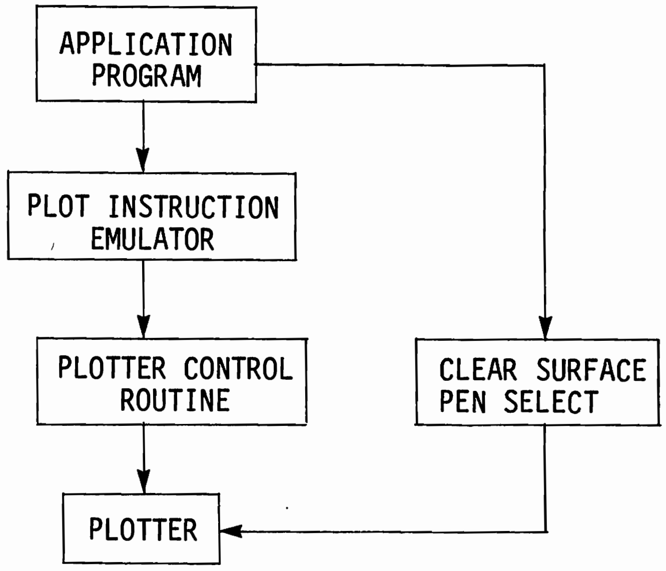


Figure 1-2. Modified Hierarchy of Plotter Control (Plot Instruction Emulator Called Directly by Application Program)

1.3 <u>USER RESPONSIBILITIES</u>

While the plotter utilities relieve the programmer of many of the tedious and often complex "housekeeping" functions involved in plotter control, they should not be regarded as independent, self-contained programs. There are several important functions associated with plotter control which must be carried out by the programmer in his application program. These user-performed functions include the following:

1. Scaling. The plotter utilities operate exclusively with plotter units. The use of absolute units enables the utilities to avoid the problem of compensating for differences between plotter models. The user must scale for his application, and compensate for his particular plotter, in his program; coordinates and distances passed to the subroutines must be in plotter units.

2. Error Checking. In general, the plotter subroutines do not check parameters passed from the application program for errors such as range errors, invalid values, etc. Such error checks must be performed by the user's program before a subroutine is called. Suggestions for specific errors which should be checked for each subroutine are provided in the "Restrictions and Exceptions" section of the chapter devoted to the subroutine.

3. Designing Special Symbols. The START module of the plotter utility package initializes either one or both of two character generation arrays. One array is called the Standard Array and contains 63 English alphanumeric characters and symbols. The second array is the Enhanced Array and contains an additional 26 special characters including Greek letters and electronic symbols. Each array is stored on the user's disk with the START module. Subsequently, the user can plot any of the characters by recalling the array and using one of the Plot Character String subroutines. If the user wishes to plot special characters not included in the standard or enhanced array, he must design his own characters according to the procedure explained in Appendix C.

4. Radian-to-degree conversion. The subroutines operate exclusively with degrees. If the programmer works in radians, he must convert from radians to degrees in the application program prior to passing angles to a subroutine. Note also that the subroutines which work with angles (the two plot character routines and the plot circle routine) automatically select degrees for their own computations. If the programmer is working in radians, he must reselect radians in his application program following execution of the subroutine.

1.4 THE "ACTIVE PLOTTING AREA"

Before a plot can be executed with the plotter utilities, the programmer must define an "active plotting area." The "active plotting area" is a rectangular region on the plotting surface within which all plotting will take place, and its limits are defined by the Set Plotter Boundaries Routine (DEFFN'19).

The active plotting area may occupy the entire physical plotting surface or any portion of it (Figure 1-3). Once defined, the boundaries of this area mark the limits beyond which no plotting is carried out.

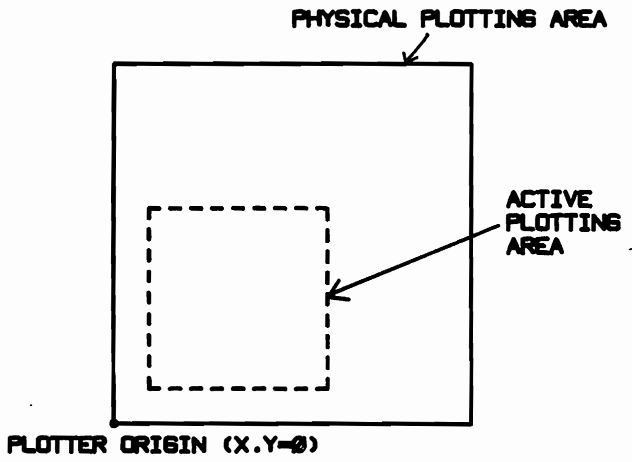


Figure 1-3a.
Active Plotting Area
Occupies a Portion of
Physical Plotting Area

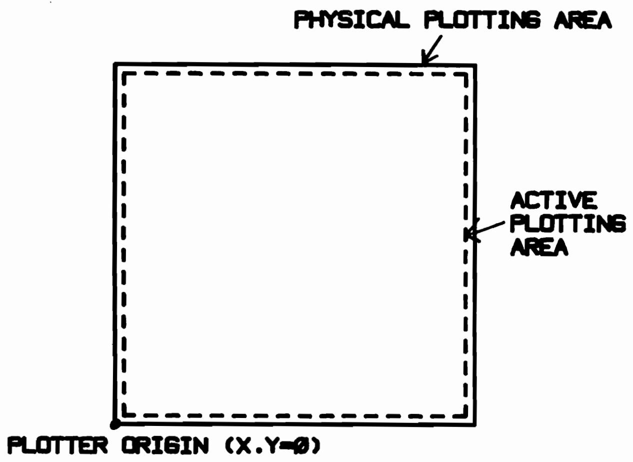


Figure 1-3b.
Active Plotting Area
Occupies All of Physical
Plotting Area

Clipping

All plotter movement beyond the boundaries of the active plotting area is inhibited by the plotter utilities. The technique of terminating a plot at a predefined boundary is known as "clipping." The plotter utilities perform automatic clipping in all cases.

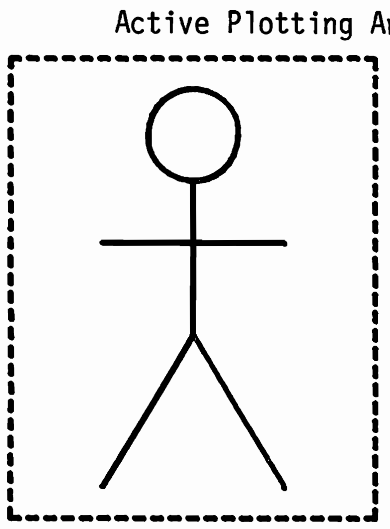


Figure 1-4a.
Plotted Figure with No Clipping (Figure Fits Completely Within Active Plotting Area)

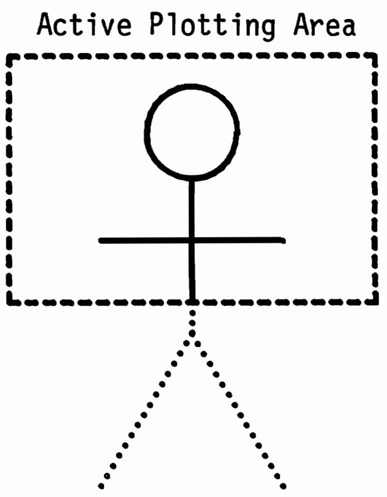


Figure 1-4b.
Plotted Figure with Clipping. Clipped Portion (Not Actually Plotted) is Indicated by Dotted Line

To avoid truncating a plot when such truncation is not desired, the programmer must see to it that the plot falls completely within the active plotting area. This problem can be minimized by defining the boundaries of the active plotting area to coincide with the physical limits of the plotting area. For many applications, however, it may be desirable to restrict plotting to a particular region of the physical plotting area, or to truncate unwanted segments of a plot. By altering the size and location of the active plotting area, the programmer can reproduce any specific portion of plot; by modifying the scaling factor along with the plotter boundaries, a specific segment of a plot can be enlarged or reduced. (See Figures 1-4, 1-5.)

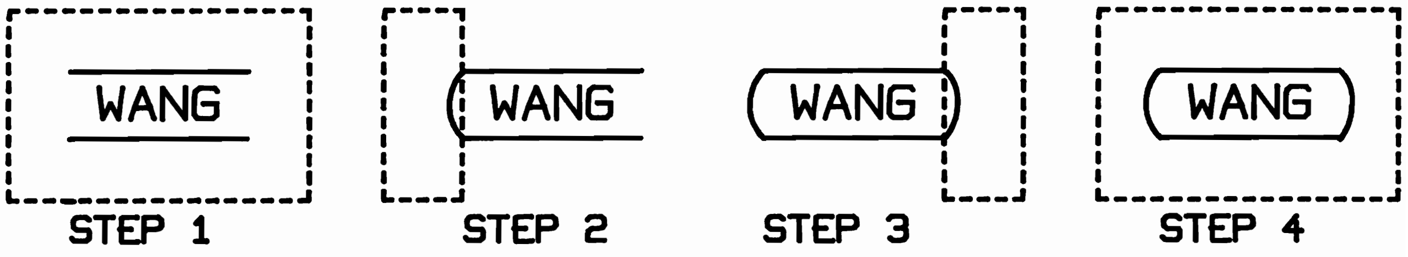


Figure 1-5. An Illustration of the Use of Clipping: Plotting the Wang Logo. The Dashed Line Represents the Perimeter of the Active Plotting Area at Each Step. The Curved Lines at Both Ends of the Logo are Produced by Clipping Circles. The Circles are Plotted with the Plot Circle Routine (DEFFN'27); by Dynamically Altering the Boundaries of the Active Plotting Area at Steps 2 and 3, the Program Causes Only the Desired Segment of Each Circle to be Plotted.

1.5 CHARACTER GENERATION

The plotter utilities provide for two sets of characters. The Standard Character Set contains 64 alphanumeric characters, including all characters found on the system keyboard, with the exception of lower case letters. The Enhanced Character Set contains upper and lowercase Greek characters and nine commonly used electronic symbols. (The characters in each set are illustrated in Appendix B.) Each character is defined by means of a 9 x 11 matrix of vertices (Figure 1-6). The vertices are numbered row by row, starting at the lower left-hand corner and concluding at the upper right-hand corner. Individual characters are defined by specifying a unique sequence of vertices between which lines are to be drawn. When the characters are plotted, the relative values of the vertices are converted to plotter coordinates, and the character is reproduced at a selected location.

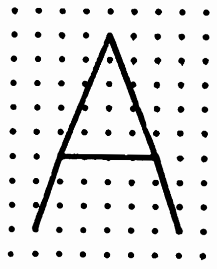


Figure 1-6. Characters Defined By Means of a Matrix of Vertices

Characters in the standard character set and in the enhanced character set are initially defined by the Initialize Character Array routine, which is accessed via the START module during system startup. The character array created by this routine is stored on disk or tape following initialization, and can be recalled into memory with the Load Character Generation Array routine (see Chapter 6). The array must be resident in memory in order to plot characters.

If the programmer wishes to plot one or more characters not provided in the two character sets, he must modify the system-generated character arrays, or else produce a completely new arrays. In either case, the process is somewhat complex and is described in Appendix C.

1.6 RESERVED VARIABLES

In general, all variable names which contain the letters A, B, C, D, E and F are reserved for use by the utility routines and should not be utilized by the programmer in his application program. Although not every variable within this range is currently used by the utilities, any additional variables required for future updates will be drawn exclusively from this group. Appendix D lists the specific variables employed by the most recent version of the utilities routines.

CHAPTER 2
SYSTEM START-UP

2.1 LOADING THE "START" MODULE

The plotter utilities must be accessed initially through the START module. The START module may be loaded directly from the Utilities disk, or it may be loaded via the Integrated Support System (ISS).

Loading the START Module Directly

If the plotter utilities are not accessed through the ISS, the START module must be loaded directly from the Plotter Utilities Disk or tape. In this case, the disk or tape must be mounted in the drive having the default address (normally 310 for disk, 10A for tape), since the START module automatically attempts to load the utilities from the default address. To load the START module itself, enter LOAD "START" (for tape) or LOAD DC F "START" (for disk), and key RETURN(EXEC).

Loading the START Module Via ISS

If the Integrated Support System is used to access the plotter utilities, the utilities disk may be mounted in any drive. The ISS APPLICATION routine permits the operator to select the device address at which the utilities disk is mounted, and the START module is automatically loaded from that address.

The START Module Menu

When the START module is executed (via RUN(EXEC)), it displays an abbreviated menu containing three items: INIT CHARACTER ARRAY, GENERATE UTILITY SET, and RETURN TO 'START'. The last routine, accessed via Special Function Key 31, is used only for ISS applications; it causes the ISS system START module to be reloaded. INIT CHARACTER ARRAY and GENERATE UTILITY SET are described in the following sections.

2.2 INITIALIZING THE CHARACTER GENERATION ARRAY

As the preceding chapter explained, the system itself will create two character generation arrays containing coded plotting routines for a total of 89 alphanumeric characters and symbols. A character array is a prerequisite for all character plotting with the Plotter Utilities. The array is initialized and stored in a data file on disk or tape with the INIT CHARACTER ARRAY routine, accessed via Special Function Key 0.

In disk versions, before the routine is loaded the operator must specify either the Standard array or the Enhanced array. Also, the operator is requested to indicate the device address of the output disk on which the character array is to be stored, and to enter the file name assigned to the character data file on disk.

In tape versions, the operator is requested to enter the device address of the output tape. After a tape has been mounted at this address, the operator must enter the number of files to be skipped prior to recording the character data file. If the character data file is to become the first file on the tape, the number of files to be skipped is zero. Otherwise, the number of files to be skipped is equal to the number of files already recorded on the output tape.

The system then opens a data file on the specified output disk or tape and stores the contents of the character array in the file. Note that since the character data file is a cataloged file, the output disk must contain a catalog.

NOTE:

The Wang BASIC Sort statements are employed by the INIT CHARACTER ARRAY routine. If your system does not offer the Sort statements, you cannot use the utilities character set. You can, however, use the built-in hardware character set provided by the plotter itself (all Wang plotters except the Model 2232B). Refer to Appendix F for further details.

2.3 LOADING THE PLOTTER UTILITIES

The GENERATE UTILITY SET routine (Special Function Key 1) permits the user to load selected plotter utilities from the utilities disk into memory. Before loading any routines, however, the operator must indicate which graphic output device is to be used:

```
ENTER THE OUTPUT DEVICE.
    1 - 2202
    2 - 2212, 2232 (2272 or 2282)
    3 - 2272 or 2282
    4 - Tektronix
    5 - 2281P
```

The response entered determines which version of the Plotter Control Routine will be loaded. Note that the Model 2272 and Model 2282 are listed next to Option 2 as well as Option 3. The control routine which supports the 2212 and 2232B (Option 2) also supports the 2272 and the 2282 and permits the same programs to be used with all four plotters. The control routine identified by Option 3 is designed exclusively for the 2272 and 2282 to provide optimized performance. This routine should be selected whenever compatibility with other plotters is not an important consideration.

If Option 4 (Tektronix Graphic Display) is selected, the operator is asked if he wants the Graphic Input (GIN) Mode Routine:

```
DO YOU WANT THE GIN MODE ROUTINE? (Y/N)
```

The GIN Mode Routine enables the System 2200 to receive graphic input in the form of screen coordinates from a Tektronix Graphic Display Terminal. Only selected terminals have the ability to transmit graphic input; however, check your Tektronix documentation to determine whether your terminal offers this feature. The GIN Mode Routine operating instructions are found in Appendix G.

Once the system has determined which Plotter Control Routine to use, it displays a menu listing nine of the eleven utility routines. (The Load Character Generation Array and Plotter Control Routines are not listed in the menu because they are automatically loaded by other routines if they are needed.)

```
S.F.   DESCRIPTION                  S.F.   DESCRIPTION

 1  -  INSTRUCTION EMULATOR          6  -  CIRCLE
 2  -  CHARACTERS (LINE)             7  -  BORDER
 3  -  CHARACTERS (ON CIRCLE)        8  -  SET PLOTTER BOUNDARIES
 4  -  LINE BETWEEN TWO POINTS       9  -  CLEAR SURFACE/PEN SELECT
 5  -  GRID
13  -  LOAD FLAGGED ROUTINES        15  -  RETURN TO 'START'
```

One or more desired routines may be selected by keying the appropriate Special Function Keys. As a Special Function Key is depressed, the system flags other routines required by the selected routine by displaying an asterisk immediately to the left of the special function number in the menu. When all desired routines have been selected, Special Function Key 13 must be depressed to load the flagged routines into memory.

Each routine automatically loads in any other routines required for its operation. For example, the Plot Instruction Emulator Routine is used by every special routine and is automatically loaded whenever any one of the other routines is selected. It need not therefore be selected by the operator. If, on the other hand, the Plot Instruction Emulator alone is wanted, it must be selected explicitly. Note that a routine is never loaded twice. If the operator selects one of the special routines (say, Set Plotter Boundaries) which automatically loads the Instruction Emulator, and also selects the Plot Instruction Emulator Routine explicitly, the Emulator Routine is loaded only once.

2.4 ALTERING LINE 30 OF THE UTILITY ROUTINES

NOTE:

Key in LISTS to perform the following alterations to line 30 of the plotter utility routine.

Line 30 contains a GOSUB'99 statement and is the same in all plotter utility routines. The GOSUB'99 statement is used by the system when loading the utilities from disk and must be altered by the programmer when all the desired routines are loaded. The specific change to be made is determined by the manner in which the subroutines will be accessed. The programmer has three options:

1. The plotter utility subroutines normally occupy the lowest line numbers in memory. If the routines are not renumbered, the GOSUB'99 statement should be changed to provide an access point to the user's program. For example, if the user's program begins at line 3000, the GOSUB'99 statement in line 30 should be changed to GOTO 3000.

2. If the plotter subroutines are renumbered to follow the user's program in memory, the GOSUB'99 statement in line 30 should be deleted.

3. If the plotter subroutines are to be accessed directly from the keyboard via Special Function Keys (refer to Section 2.5), the GOSUB'99 statement in line 30 should be replaced with a STOP.

2.5 ACCESSING THE PLOTTER UTILITIES FROM THE KEYBOARD VIA SPECIAL FUNCTION KEYS

The plotter utilities are written as marked subroutines (DEFFN'), and are normally called from a user's program. However, the integers used to identify the subroutines all fall within the range 0-31, and each subroutine may therefore also be called directly from the keyboard with one of the Special Function Keys 0-31.

Once the desired routines have been loaded into memory, they must be RUN in order to resolve the program and allocate variable space (the GOSUB'99 at line 30 must be replaced with a STOP). At that point, any resident subroutine can be accessed in immediate mode by keying the required parameters into the display (individual parameters must be entered on one line and must be separated by commas) and depressing the Special Function Key associated with the desired routine. Note that the plotter utility routines all have numbers greater than 15, and therefore employ the uppercase Special Function Keys (requiring that the SHIFT key be down when the Special Function Key is depressed). The parameters are passed directly to the subroutine, which immediately proceeds to execute a plotter operation after the last parameter is accepted. A complete list of the parameters which must be passed to each subroutine, and the order in which they must be specified, is included in the chapter devoted to each routine and also in Appendix A.

2.6 CLEARING THE 'START' MODULE

Whether the group of selected routines loaded with the START module is to be incorporated into the user's application program, or saved as a separate program file which can be conveniently recalled as needed, the START routine itself should first be cleared. The START routine occupies lines 9800 through 9980 inclusive and is not wiped out when the utility routines are loaded into memory. The START routine can be cleared by executing the following statement:

```
CLEAR P 9800
```

This statement causes all program lines from line 9800 on to be cleared from memory.

NOTE:

It is recommended that the START routine be cleared immediately after the selected plotter utilities are loaded.

CHAPTER 3
SET PLOTTER BOUNDARIES (DEFFN′ 19)

3.1 PROGRAM DESCRIPTION

The Set Plotter Boundaries Routine is used to initialize, and subsequently to reset, the limits of the active plotting area. The boundaries of the active plotting area circumscribe the area in which all plotting will take place; lines which extend beyond a boundary are clipped at the boundary. The Set Plotter Boundaries routine requires the coordinates of two reference points, one at the lower left-hand corner of the active plotting area, and the other at the upper right-hand corner. These points are used to construct a rectangle which defines the active plotting area (Figure 3-1).

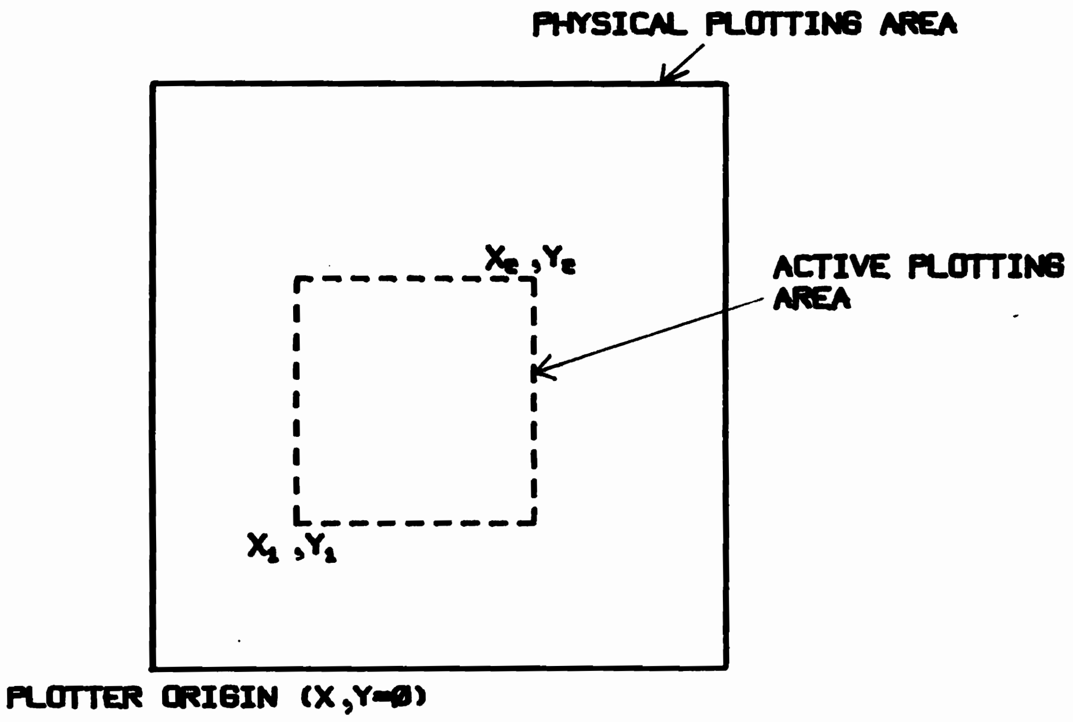


Figure 3-1. Two Points Used to Define the Active Plotting Area

3.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'19 (A8,B8,A9,B9,D4)
```

Where:

A8 = X Coordinate, Lower Left Corner.
B8 = Y Coordinate, Lower Left Corner.
A9 = X Coordinate, Upper Right Corner.
B9 = Y Coordinate, Upper Right Corner.
D4 = Option Code.

0 - Reset Boundaries (No Plotter Movement).
1 - Initialize Plotter Boundaries(Move Plotter to Origin, then to Lower Left).
2 - Reset Boundaries (Move Plotter to Lower Left).

NOTES:

1. The variables shown (A8, B8, A9, B9, D4) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.

2. All coordinates must be specified in plotter units relative to plotter origin.

X and Y Coordinates, Option Code

In addition to the X and Y coordinates of the two reference points, the subroutine must be passed with an Option Code which identifies the operation to be performed. There are three options available: initialize plotter boundaries, reset boundaries and move plotter home, and reset boundaries with no plotter move. Each option is identified by one of the Option Codes 0, 1, or 2.

Option 1 is used to initialize the boundaries of the active plotting area immediately after system startup. The boundaries must be initialized with Option 1 before any other plotting subroutines are called. (Note: On a Tektronix graphic display, Option 1 clears the display in addition to setting the boundaries.)

Options 0 and 2 are used to reset existing plotter boundaries after the boundaries have been initialized with Option 1. Option 0 causes the boundaries to be reset without moving the plotter from its current position. Option 2 causes the boundaries to be reset and moves the plotter from its current position to the lower left-hand corner of the newly-defined plotting area. (Note: On a Tektronix graphic display, Options 0 and 2 reset boundaries without altering the display image.)

Note that although the boundaries of the active plotting area circumscribe the area in which plotting will take place, the lower left-hand corner of the active plotting area is not interpreted as the origin point with respect to which coordinates are expressed. All coordinates are interpreted as relative to the absolute plotter origin 0,0. If the lower left-hand corner of the active plotting area does not coincide with the plotter origin, it must be understood that points will be plotted with respect to the plotter origin. For example, if the coordinates of the lower left are 100, 100, and the plotter is subsequently instructed to plot a point at 50, 50, the point is not plotted, because it does not fall within the active plotting area.

SPECIAL NOTE CONCERNING THE MODEL 2202

The Y axis of the Model 2202 Plotting Output Writer extends infinitely in both the positive and negative directions. For this reason, the Model 2202 has no "absolute" plotter origin, since although X = 0 at the left-hand margin, Y *may* arbitrarily be assigned a value of zero at any point on the vertical axis. When an Option Code of 1 is issued on the 2202, therefore, it instructs the plotter to issue a Carriage Return which moves the carriage from its current position to the left-hand margin, without performing a line feed. This point is then defined as the plotter origin point (X,Y = 0) for subsequent plotting operations.

All subsequent plotter coordinates, including the coordinates of the active plotting area as well as coordinates subsequently passed to any plotter utility routine, are interpreted as relative to the defined plotter origin point.

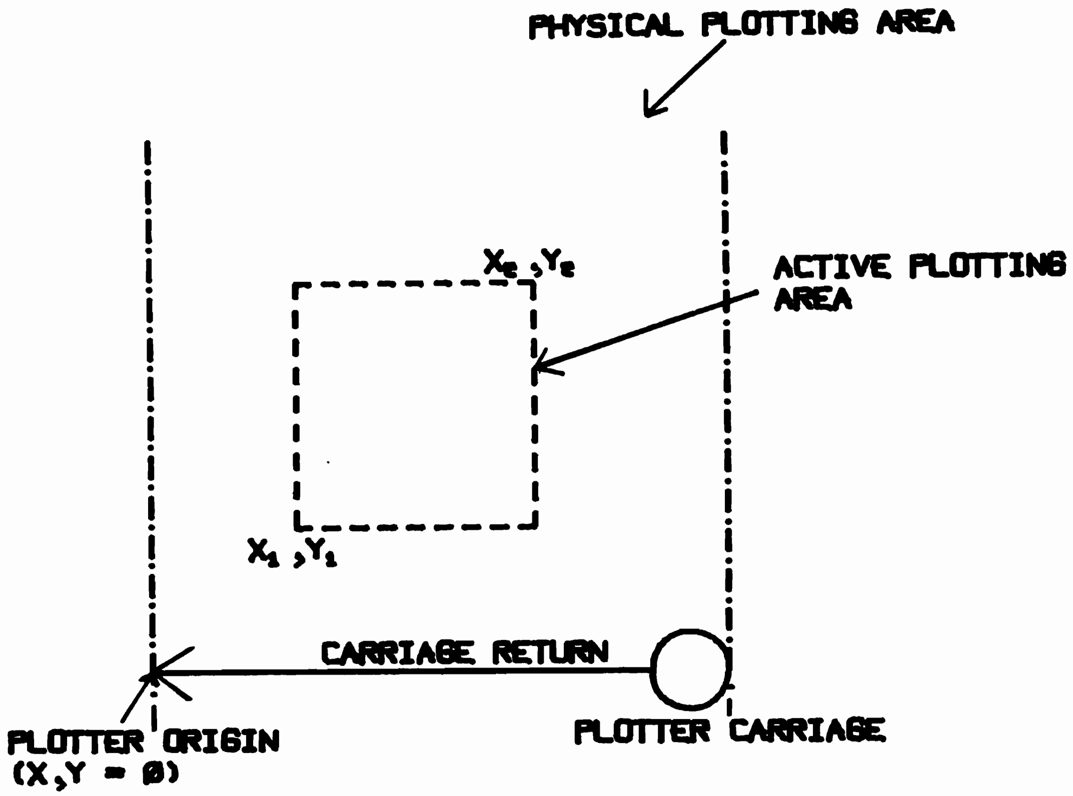


Figure 3-2. Defining Plotter Origin and Initializing Active Plotting Area on Model 2202

If the boundaries of the active plotting area are subsequently altered with Option 0 or Option 2, the new coordinates are relative to the plotter origin defined by Option 1. When, however, a second Option 1 code is issued, the system issues a carriage return and defines a new plotter origin at that point. This situation is to be avoided when plotting, since it has the effect of wiping out the existing reference point and defining a new one. Option Code 1 is therefore used only to set up the plotter boundaries initially; any subsequent alteration of the boundaries must be done with Option 0 or 2.

SPECIAL NOTE CONCERNING THE MODEL 2272

The Model 2272 plotter permits the operator to change the orientation of the X and Y axes with the AXIS SELECTION Switches. This is done independently of the Plotter Utilities, and generally has no effect upon the operation of the utility routines themselves. When rotating a non-symmetrical plot, however, care must be taken to avoid extending a part of the plot beyond the boundaries of the plotting area.

The Model 2272 also permits the user to set the limits of the physical plotting area with the SET LIMITS switch. A line which extends beyond the defined limits is automatically "clipped" by the plotter itself. The clipping performed by the plotter is much more elementary than the clipping performed by the Plotter Utilities, however, since the plotter performs no interpolation. Thus, a line which extends beyond the plotter limits is not plotted at all by the plotter, while the Plotter Utilities, in contrast, plot any portion of a line which falls within the active plotting area. To ensure that the portion of a line which falls within the active plotting area will be plotted, therefore, always see to it that the active plotting area defined with DEFFN'19 falls completely within the physical plotting area defined with the plotter's SET LIMITS switch.

In general, the lower left of the active plotting area will be defined to coincide with the origin of the physical plotting area. On the Model 2272, the origin (home) position can be defined anywhere on the plotting surface, either manually or under program control. A suggested procedure for resetting the home position and initializing the active plotting area so that the lower left corresponds to the origin point is described below:

1. Manually position plotter to desired home position with SLEW switch.

2. Reset home position under program control with HEX(E4) code, which defines home position at current plotter location. For example:

   ```
   PLOT <,,HEX(E4)>
   ```

   This operation automatically sets the limits of the physical plotting area to the default limits. On the X-axis, the default limits are the left and right borders of the plotter. On the Y-axis, the default limits are ± 8191 plotter units from the home position.

3. Set limits of active plotting area with a GOSUB'19 subroutine call. For example:

   ```
   GOSUB'19 (0,0,X,Y,1)
   ```

   where the 0,0 causes the lower left corner to be set at the home position, and variables X and Y contain the coordinates of the upper right corner.

3.3 RESTRICTIONS AND EXCEPTIONS

1. The X coordinates of the lower left corner must be smaller than the X coordinate of the upper right corner. Similarly, the Y coordinate of the lower left must be smaller than the Y coordinate of the upper right. If the lower left coordinates exceed those of the upper right in either case, the coordinates will be accepted by the set plotter boundaries routine, but no subsequent plotting is possible.

2. The boundaries of the active plotting area should not be located outside the physical plotting area (i.e., the physical screen or plotter bed). This error results from the specification of coordinates which lie outside the limits of the physical plotting area. The reasons for avoiding such an error may be obvious. The plotter subroutines maintain internal pointers which identify the plotter's position at all times. If the plotter is inadvertently sent to a point outside the physical plotting area, the plotter itself must stop at the edge of the plotter bed. The plotter subroutine, however, recognizes only the boundaries of the active plotting area as limits, and will update the plotter position to the specified point outside the plotter bed, so long as this point is within the active plotting area. The end result of such an operation is that the subroutine's internal pointers no longer accurately identify the physical position of the plotter. In effect, the controlling subroutine loses track of where the plotter is. This situation creates problems for all subsequent plotting. Note that the whole problem is obviated if the active plotting area is defined within the physical plotting area. In that case, all plotter movement is terminated at the boundaries of the active plotting area, and the disparity between physical plotter position and assumed plotter position never arises.

3.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. `100 GOSUB'19 (0,0,M*400, N*400,1)`

2. `50 GOSUB'19 (M,K,H+H*400, K+K*400,1)`

3. `200 GOSUB'19 (M,N,M1,M2,0)`

4. `150 GOSUB'19 (P-N*100,P1-N*100,P+N1*100,P1+N1*100,2)`

CHAPTER 4
PLOT CHARACTER STRING (STRAIGHT LINE) (DEFFN′ 20)

4.1 PROGRAM DESCRIPTION

The Plot Character String (Straight Line) routine plots a specified string of alphanumeric characters on a straight line. The characters plotted by this routine are defined in a Character Generation Array, *which must be* resident in memory when the routine is run (see Chapter 6, "Load Character Generation Array").

The character string is plotted on a straight line called the "character base line." A single reference point whose coordinates are passed by the programmer is used to define the base line. The line itself is not plotted, however. Options permit the programmer to define the character size, and to slant the characters and rotate the character string relative to the horizontal axis.

4.2 SUBROUTINE ARGUMENT LIST

DEFFN'20 (C$, C, C6, C7, C1, C2, C3)

Where:

C$	=	Character string to be plotted.
C	=	Character size.
C6	=	Slant reference angle (degrees).
C7	=	Rotation angle (degrees).
C1	=	X coordinate of reference point.
C2	=	Y coordinate of reference point.
C3	=	Option Code.

0	=	Start at reference point.
1	=	Center at reference point.
2	=	End at reference point.

NOTES:

1. The variables shown (C$, C, C6, C7, C1, C2, C3) are reserved for use by the subroutine, and should not be used by the programmer in a GOSUB' statement.

2. All coordinates must be expressed in plotter units, relative to the plotter origin.

3. Angles must be expressed in degrees.

4. The BASIC language delineators (,;) must be within quotes to be plotted (e.g., "1,2").

Character String

The first argument passed is the character string to be plotted. It may contain any keyboard character, may be a maximum of 64 characters in length, and may be specified as a literal string in quotes or as the value of an alphanumeric variable. (The programmer also can elect to define his own special characters; see Appendix C for a discussion of this procedure.) Embedded spaces in the character string are treated as part of the string, but trailing spaces (even when enclosed in quotes) are ignored.

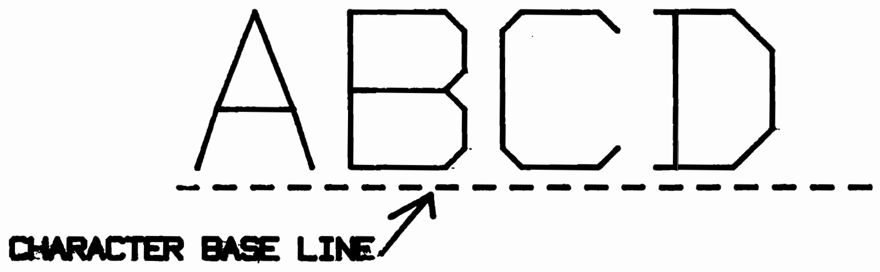


Figure 4-1. A Plotted Character String, Showing the Character Base Line (Note: Base Line is Not Actually Plotted)

Character Size

The character size is specified by the programmer with the second argument passed to the subroutine. Any valid numeric expression may be used. A 9 x 11 matrix of vertices is used to define each character, and the character is plotted by connecting a specified sequence of vertices with straight line segments (Figure 4-2). Note that the full 9 x 11 matrix is not used in defining the character; instead, the character is defined with a 7 x 9 matrix embedded within the larger matrix. For this reason, each character is surrounded on all four sides with a "buffer" equal in size to the distance between two consecutive vertices. The buffer is used to maintain proper spacing between consecutive characters in a character string.

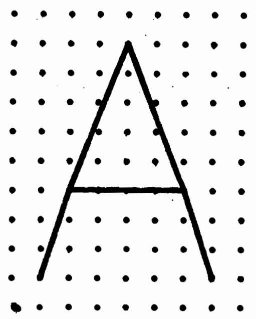


Figure 4-2. The 9 x 11 Matrix of Vertices Used in Defining a Character

Character size is determined by the distance between neighboring vertices in the defining matrix. In a character of unit size (size = 1), the distance between neighboring vertices is one plotter increment. Thus a unit character is six plotter increments in width, and eight plotter increments in height, and is surrounded by a one-increment buffer on all four sides. Because the length of a plotting increment differs among different plotter models, the actual dimensions of a unit character also differ. On the Model 2212, which has a plotting increment of .01 inch (at full scale), the unit character is .05 inch x .07 inch in size. On the Model 2232B, with a plotting increment of .0025 inch, the unit character is one-fourth as large, .0125 inch x .0175 inch.

Because the unit character is much too small for most purposes, the routine permits selection of a larger character size. The "character size" is defined as an integer multiple of the unit size. Thus, a character size of 2 produces a character twice the size of the unit character; a size of 10 produces a character ten times as large as the unit character, etc. It is recommended that the expression used to specify character size be an integer value.

For the subroutine's purposes, the character size is interpreted as the number of plotter increments between neighboring vertices in the defining matrix. Thus, in the matrix used to define a character of size 2, neighboring vertices are two increments apart; in the matrix for a character of size 10, the interval between neighboring vertices is ten increments, etc. (refer to Figure 4-3). Characters of different sizes produced on each of the plotter models are illustrated in Appendix B.

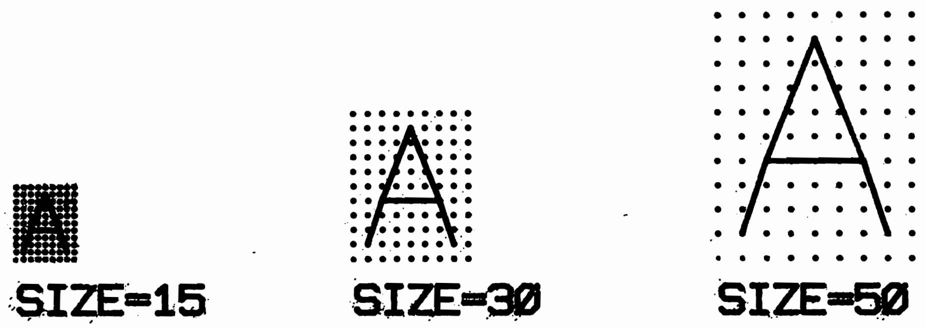


Figure 4-3. Defining Matrices for Characters of Different Sizes (Plotted on Model 2232B)

It must be emphasized that the distances between all neighboring vertices in the defining matrix are enlarged as the character size increases. The size of the buffer surrounding each character therefore increases proportionately with the character size. This fact has several implications. It implies, firstly, that the distance separating consecutive characters will increase as the character size increases. A character of size 2, for example, is surrounded by a buffer of two plotter increments. Since the next consecutive character also will have a two-increment buffer, the total distance between two consecutive characters is four plotter increments (see Figure 4-4). The buffer around each character therefore ensures that the spacing between characters in a string always remains proportional to the character size.

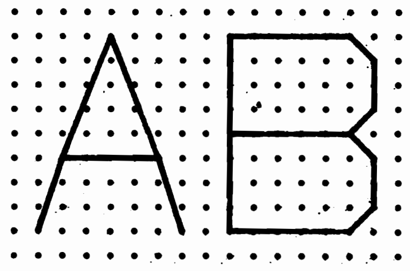


Figure 4-4. Spacing Between Consecutive Characters Proportional to Character Size.

A second implication of the proportional increase in buffer size is that the distance between the base of a character and the character base line increases as the character size increases. This is the case because the base of the defining matrix, and not the base of the character itself, rests on the base line; the buffer therefore intervenes between the base of the character and the base line. Thus, the base of a character of size 2 is two increments above the base line; the base of a size 10 character is ten increments above the base line, etc. For the same reason, the first character in the string is displaced to the right of the starting point a distance which is proportional to the character size. This can be seen if a perpendicular is drawn to the base line at the starting point (Figure 4-5). The proportional displacement of the character string above the base line and to the right of the starting point must be considered when accurate placement of the character string is critical, since it is the position of the character base line, and not that of the character string itself, which must be specified by the programmer.

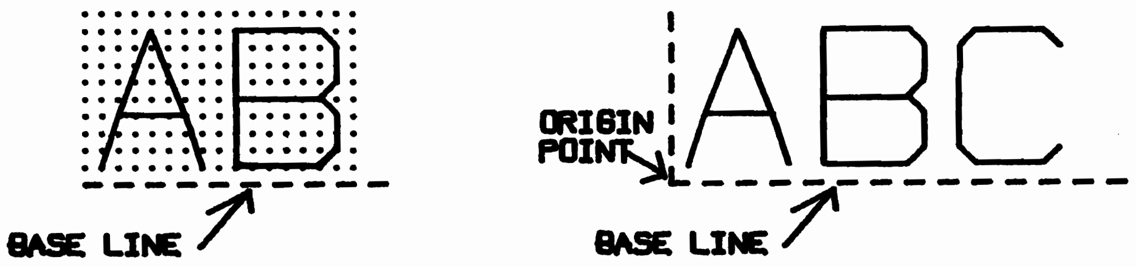


Figure 4-5. Position of Character String Relative to Starting Point and Character Base Line.

The Plot Character String routine will accept a negative value for the character size. A negative character size causes the defining matrix to be inverted on both the horizontal and vertical axes. The plotted result is a character of the specified size which is both upside down and backward (see Figure 4-6). This feature has no obvious application when plotting characters in a straight line; it can be useful, however, when plotting characters on a circle. (Characters on the bottom half of the circle normally plot upside down and backward; a negative size in this case therefore causes the characters to plot correctly.) Refer to Chapter 5 for a discussion of plotting characters on a circle.

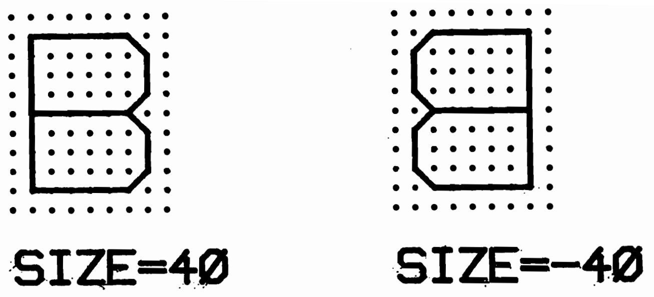


Figure 4-6. Positive and Negative Character Size

Character Slant

"Character Slant" is achieved by displacing the X coordinates of vertices in the defining matrix to the right or left of a line perpendicular to the base line, without changing the Y coordinates. (See Figure 4-7.) The effect of such a displacement is to produce a character or string of characters which slant relative to a line perpendicular to the character base line. A positive displacement produces a slant to the right of the perpendicular, while a negative displacement produces a slant to the left (Figure 4-8).

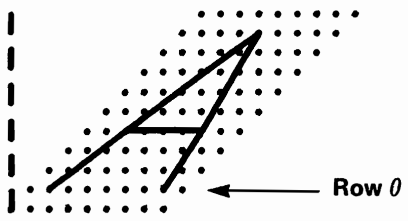


Figure 4-7. Character Slant Produced by Displacement on X-Axis of Vertices in Defining Matrix

The displacement of vertices in row 0 of the matrix is always zero. In each successive row, the displacement of the X coordinate is proportional to the Y coordinate of that row. The degree of slant is therefore determined by the amount of displacement in each row of the matrix, and this is computed on the basis of a "slant reference angle" specified by the programmer. The slant reference angle must be passed in degrees. It is used to compute a displacement for each vertex in the matrix according to the formula:

$D = Y * \text{SIN}(\Theta)$

where D is the displacement, Y is the Y coordinate of the vertex, and Θ is the slant reference angle specified by the programmer. The displacement computed by this formula is added to the X coordinate of the vertex:

$X = X + D$

Clearly, if is a positive angle, and thus D is a positive value, the new X coordinate will be located to the right of the original; if D is negative, the new coordinate will be to the left of the original.

This technique ensures that the character's displacement from the perpendicular is proportional to its height at each vertex, producing a uniform slant relative to the perpendicular (Figure 4-8).

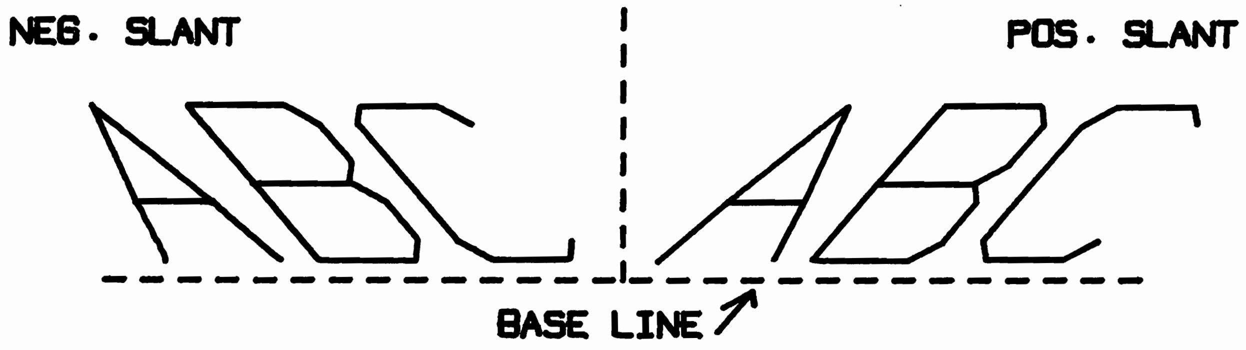


Figure 4-8. Positive and Negative Character Slant

In general, the primary consideration when plotting characters is one of aesthetics, and the programmer must simply experiment with different reference angles until he arrives at a slant which has the most pleasing effect. In rare cases, however, it may be useful to generate characters with a specific "angle of slant." The "angle of slant" may be defined as the angle formed between the vertical columns of the defining matrix and the perpendicular. When the angle of slant is zero, these columns are parallel to the perpendicular. As the slant reference angle increases from 0, the angle of slant also increases (Figure 4-9).

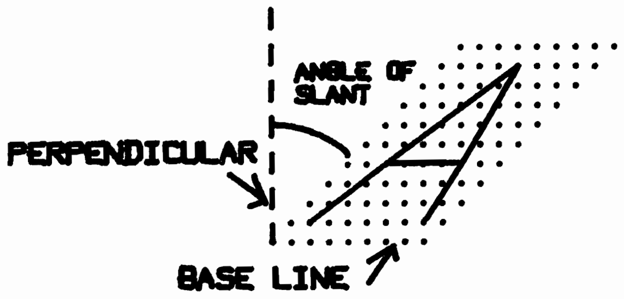
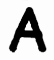
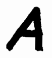
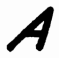
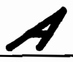
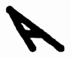
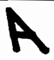
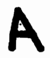


Figure 4-9. Angle of Slant

It should be evident that the angle of slant and the slant reference angle (specified by the programmer) are not equivalent; Table 4-1 illustrates the reference angles used to produce some common angles of slant. Note that, because the sine of the reference angle is used in computing displacement, the meaningful range of reference angles is restricted to between -90 and +90, inclusive.

Table 4-1. Correlations Between Slant Reference Angles and Resultant Angles of Slant

Reference Angle (Passed to Routine)	Angle of Slant (Actual Angle Between Vertical Columns and Perpendicular)	Example of Plotted Character
0	0	
15	15	
40	30	
90	45	
-90	-45	
-40	-30	
-15	-15	

Character Rotation

Character rotation is the angular distance between the character base line and the plotter X axis (horizontal axis). (See Figure 4-10.) If the angle of rotation is zero, the character base line is parallel to the horizontal axis. The angle of rotation is specified by the programmer as the fourth argument passed to the plot character subroutine. An angle of rotation between +1 and +180 inclusive causes the character base line to be rotated above the horizontal; an angle between -1 and -180 inclusive results in rotation below the horizontal. The angle of rotation must be expressed in degrees.

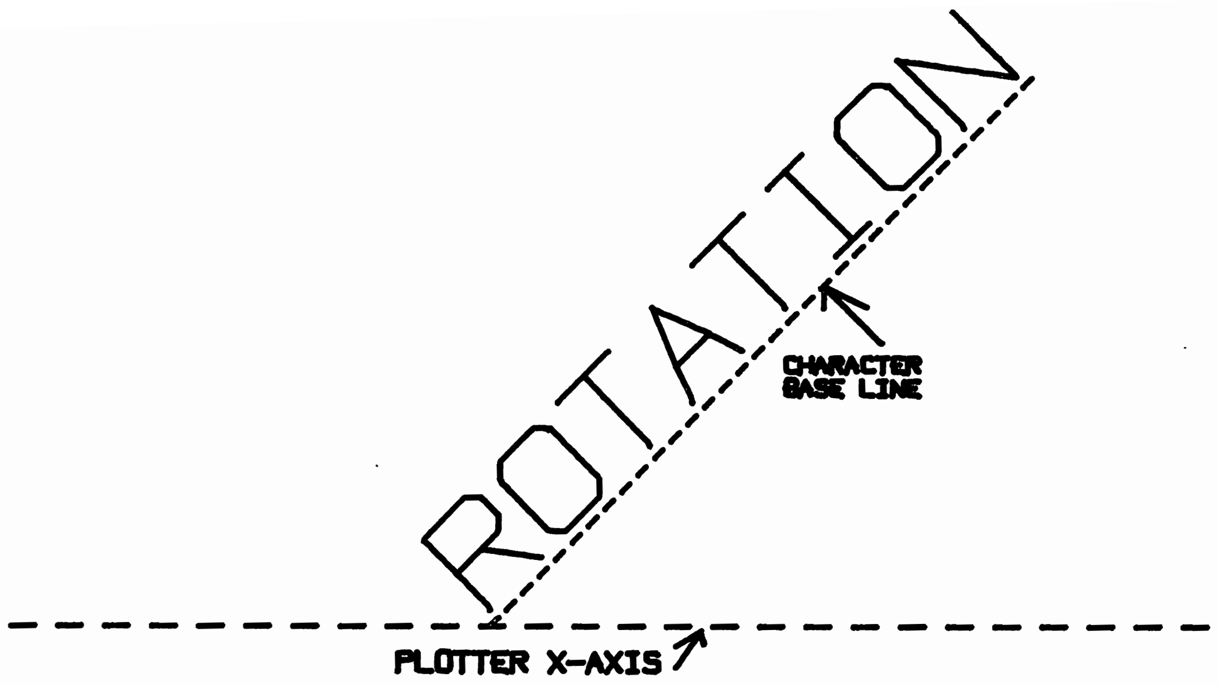


Figure 4-10. Rotation of the Character Base Line.

Note that character slant and character rotation are independent of one another. Character slant is always relative to a perpendicular on the character base line, while character rotation results in altering the orientation of the base line itself. When non-zero values are specified for both slant and rotation, the subroutine first computes the slant relative to the base line, and then rotates the base line relative to the plotter X axis (see Figure 4-11).

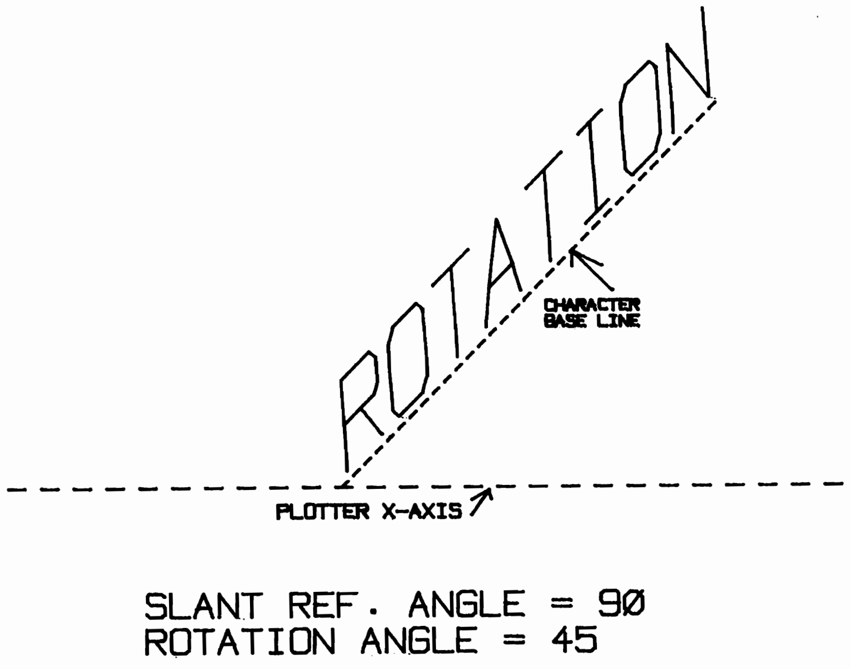


Figure 4-11. Character String Having Both Slant and Rotation

Reference Point of Character Base Line

The location of the character base line is defined with the coordinates of a single point on the line, called the "reference point."

The reference point must be either the starting point, the center point, or the ending point of the character base line. An Option Code permits the programmer to indicate to the subroutine which of the three points is specified.

The X and Y coordinates of the base line reference point are the fifth and sixth arguments passed to the subroutine, respectively. The coordinates must be expressed in plotter increments, and they must be relative to the plotter origin. (For example, the values 400, 500 define a point 400 plotter increments from the plotter origin on the X axis, and 500 increments from the origin on the Y axis.) Any legal numeric expression may be used to specify the X and Y coordinates. If a non-integer value is used to specify a coordinate, the decimal portion is automatically truncated.

In every case, the character string is plotted relative to the starting point of the character base line. If the reference point coordinates specified are designated as those of the center point or end point rather than the starting point, the subroutine itself first computes the coordinates of the starting point, and then plots the character string relative to this point. Remember, however, that due to the presence of the surrounding buffer, characters do not actually rest on the base line, nor does the character string actually start at the starting point (refer to the discussion of "character size").

Option Code

An Option Code is the last argument passed to the subroutine. This code instructs the subroutine to interpret the reference point coordinates as those of the starting point, center point, or ending point of the character base line. Three codes are therefore available:

0 -	Start at reference point.
1 -	Center at reference point.
2 -	End at reference point.

4.3 RESTRICTIONS AND EXCEPTIONS

1. The Character Generation Array must be resident in memory when this routine is run. If the array is not resident when DEFFN'20 is called, all characters are plotted as spaces.

2. A character size of 0 is accepted by the subroutine without an error. The plotter will move to the starting point of the character string and remain there without plotting while all internal operations associated with plotting the string are carried out, at which point control is returned from the DEFFN'20 subroutine to the application program.

3. Characters, or portions of characters, which extend beyond the boundaries of the active plotting area are clipped at the boundary.

4. Trailing spaces are not plotted by the subroutine, even when enclosed in quotes. For example, the character string "ABCD " is plotted as "ABCD". In rare cases, it may be convenient to plot one or more trailing spaces. In these instances, a trailing space can be forced by specifying a non-plottable character at the end of the string. (All non-plottable characters plot as spaces.) Since all keyboard characters except the lowercase letters are defined in the character generation array, the only non-plottable keyboard characters are the lowercase letters a-z. Any one of these letters can be used to force a trailing space. For example, the string "ABCDe" will plot as "ABCD ".

5. The subroutine automatically selects degrees. If the programmer is working in radians, he must reselect radians in his application program immediately following subroutine execution.

4.4 <u>EXAMPLES OF VALID GOSUB' SYNTAX</u>

1. GOSUB'20 ("X-AXIS", 10,0,0,600,400,0)

2. GOSUB'20 (F$,8,90,0,M*400, N*400, 1)

3. GOSUB'20 (G$, K, P1, P2, M*M1, N*N1, 2)

CHAPTER 5
PLOT CHARACTER STRING (CIRCLE) (DEFFN′ 21)

5.1 PROGRAM DESCRIPTION

The Plot Character String (Circle) Routine plots an alphanumeric character string on the circumference of a circle whose center coordinates and radius length are specified by the programmer. The circle itself is not plotted by this routine. The characters are defined in a character generation array, which is created during system startup, and must be resident in memory when the character string is plotted. (See Chapter 6, "Load Character Generation Array".)

5.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'21 (E$, E1, E2, E3, E4, E5, E6, E7)
```

Where:

```
E$ = Character string to be plotted.
E1 = Size.
E2 = Slant Reference Angle (degrees).
E3 = X coordinate of circle center.
E4 = Y coordinate of circle center.
E5 = Radius of circle.
E6 = Reference point on circle (degree).
E7 = Option Code.
    0 = Start at reference point.
    1 = Center at reference point.
    2 = End at reference point.
```

NOTES:

1. The variables shown (E$, E1, E2, E3, E4, E5, E6, E7) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.
2. All coordinates must be expressed in plotter units relative to plotter origin.
3. The radius length must be expressed in plotter units.
4. The slant reference angle and reference point must be specified in degrees.

Character Size, Slant

The discussions of character size and slant in Chapter 4 apply to characters plotted on a circle. If the character size is positive, the characters are plotted in a clockwise direction on the outside of the circle. If, however, a negative character size is specified, the characters are plotted in a counterclockwise direction, on the inside of the circle. In either case, the base of the character matrix rests on the circumference of the circle (see Figure 5-2).

X-Y Coordinates of Circle Center, Length of Radius

The coordinates of the circle center point and length of the radius must be specified in plotter units (any valid numeric expression may be used). Characters are plotted so that the bottom of the character matrix (and not the base of the character itself) rests on the circumference of the defined circle. The characters may lie outside the circle (if the character size is positive) or inside the circle (if the character size is negative).

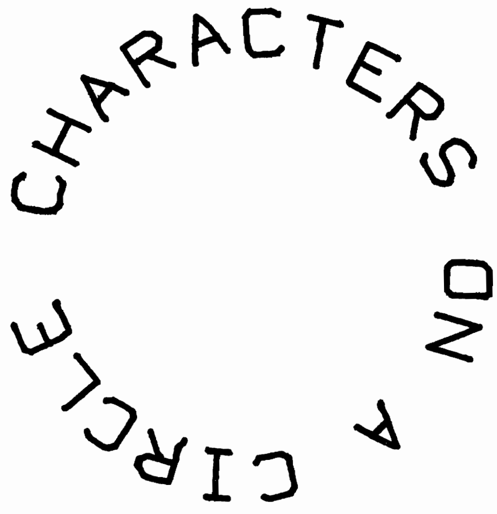


Figure 5-1. Characters Plotted on a Circle

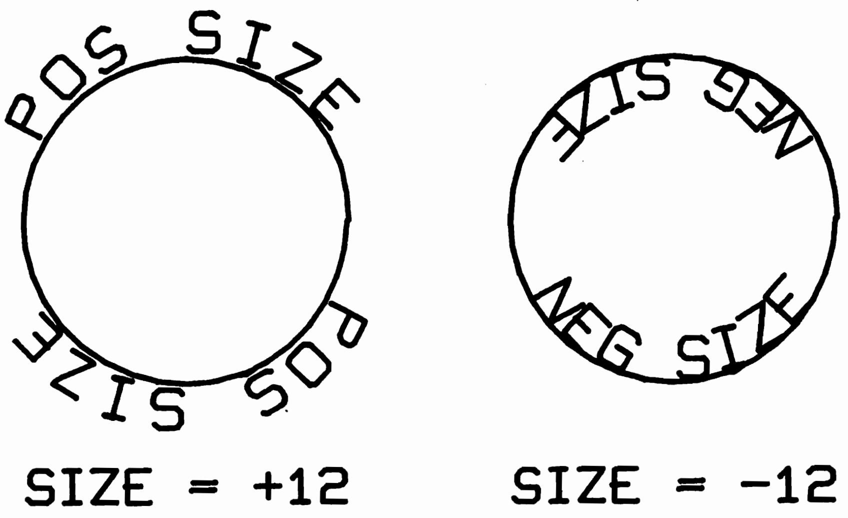


Figure 5-2. Positive and Negative Character Size

Reference Point on Circumference of Circle

The character string is plotted with respect to a reference point on the circumference of the defined circle. An Option Code permits the programmer to indicate whether the reference point will be interpreted as the starting, ending, or center point of the character string.

The reference point is specified as a degree between 0 and 360 on the circumference of the circle. Figure 5-3 below illustrates the numbering scheme for degrees on a circle.

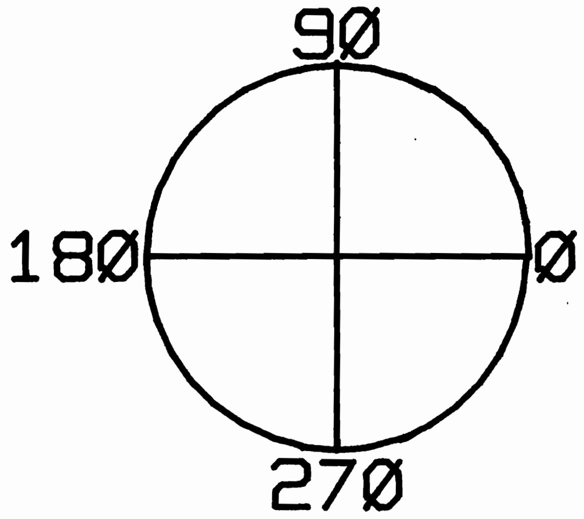


Figure 5-3. Degrees on Circumference of a Circle

Option Code

Three Option Codes permit the programmer to designate the reference point as the starting, ending, or center point of the character string:

0 - Start at reference point.
1 - Center at reference point.
2 - End at reference point.

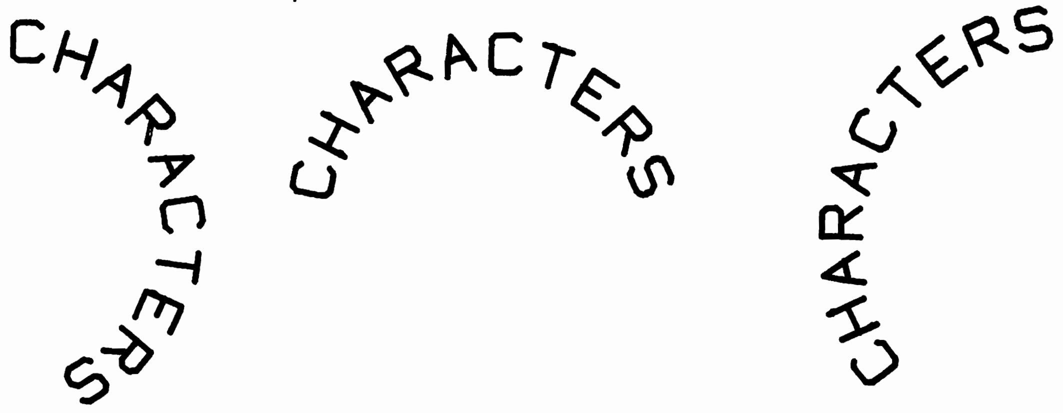


Figure 5-4. Character String Starting, Ending and Centered at 90

5.3 RESTRICTIONS AND EXCEPTIONS

1. The character generation array must be resident in memory when this routine is called. If the array is not resident when DEFFN'21 is called, the plotter plots only spaces.

2. A character size of 0 is accepted without an error indication, and causes the plotter to move to the starting point of the character string and remain there without plotting until all internal calculations required for plotting the string are completed.

3. The programmer must ensure that the total length of the character string to be plotted does not exceed the circumference of the circle. If it does, the string is "wrapped around," and the end of the string is plotted over the beginning of the string.

4. The subroutine automatically selects degrees. If the programmer is working in radians, he must reselect radians in his main program immediately following subroutine execution.

5.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. 500 GOSUB'21 ("POINT #1", 10, 0, M, N, K, L, 1)

2. 200 GOSUB'21 (F$,8,0, M3*400, M4*400, K*400, 180, 0)

3. 250 GOSUB'21 ("Y-AXIS", L1, L2, P1*100, P2*100, P7*100, L,N)

CHAPTER 6
LOAD CHARACTER GENERATION ARRAY (DEFFN '22)

6.1 PROGRAM DESCRIPTION

The Load Character Generation Array routine loads a previously-initialized character data file from a specified disk platter or tape into the receiving character generation array in memory. The character generation array (Standard or Enhanced) is initially created during system startup (see Chapter 2), and stored on tape or disk; it must be resident in memory in order to plot characters on a straight line (Chapter 4), or on a circle (Chapter 5). One file can be created for the Standard Character Generation Array, and a second file for the Enhanced Character Generation Array. The choice of which file is to be called is specified in the parameter of DEFFN'22. The procedure for loading a character array into memory is somewhat different for disk and tape versions of the Utilities.

In tape versions, the character array (Standard Array only) is loaded by executing a GOSUB'22 statement with no arguments. The data tape containing the character array must be mounted at the default tape address. The Load Character Array Routine automatically searches the tape for the character data file and loads this file into array C$() in memory.

In disk versions, the file name assigned to the character data file during system startup must be specified when the file is accessed with DEFFN'22. In addition, the disk address at which the disk containing the data file is mounted must be assigned to a file number, and this file number must be specified. For example, if the disk containing the character data file is mounted at address 310, a statement of the form SELECT #n 310 must be executed to assign one of the available file numbers 0-6 to address 310 (e.g., SELECT #1 310). The file number assigned to this address must be passed to the subroutine along with the file name. The subroutine opens the named file, and loads its contents into array C$() in memory.

6.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'22 (C$,C)
```

Where:

C$ = File name of character data file (disk version only).

C = File number (0-6) assigned to address of platter containing character data file (disk version only).

> NOTES:
>
> The variables shown (C$,C) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' Statement.

6.3 RESTRICTIONS AND EXCEPTIONS

None.

6.4 EXAMPLES OF VALID GOSUB' SYNTAX

Disk

1.
```
100 GOSUB'22 ("CHARFILE",1)
```

2.
```
100 A$ = "FILE 1"
110 SELECT #2 B10
120 GOSUB'22 (A$,2)
```

Tape

1.
```
200 GOSUB'22
```

CHAPTER 7
PLOT LINE BETWEEN TWO POINTS (DEFFN '25)

7.1 PROGRAM DESCRIPTION

The Plot Line Between Two Points routine plots a straight line between two defined points. The line may be solid, dashed, dotted, or dashed/dotted. If the line is dashed or dotted, the length of the dash, or the distance between consecutive dots, must be specified by the programmer.

7.2 SUBROUTINE ARGUMENT LIST

DEFFN'25 (F, F0, F1, F2, F3, F4)

Where:

F	=	X coordinate of starting point.
F0	=	Y coordinate of starting point.
F1	=	X coordinate of ending point.
F2	=	Y coordinate of ending point.
F3	=	Length of segment or interval.
F4	=	Option Code. 0 = Dashed line. 1 = Dotted line. 2 = Dashed/Dotted line. 3 = Solid Line (argument F3 ignored).

NOTES:

1. The variables shown (F, F0, F1, F2, F3, F4) are reserved for use by the subroutine, and should not be used by the programmer in a GOSUB' statement.

2. All coordinates must be expressed in plotter units, relative to the plotter origin.

3. The segment/interval length must be expressed in plotter units.

Coordinates of Starting, Ending Points

The X and Y coordinates of the starting and ending points of the line must be specified in plotter units. (Any valid numeric expressions may be used to specify these coordinates.) If the coordinates of either point or both points fall outside the boundaries of the active plotting area, only that portion of the line which falls within the active plotting area will be plotted.

Length of Segment or Interval (Dashed, Dotted, Dashed/Dotted Options)

The segment/interval length represents the length of line segments (dashes) for dashed lines, and the length of intervals between consecutive dots for dotted lines. This length must be expressed in plotter units (any valid numeric expression may be used). For dashed lines, the length of the interval between dashes is equal to the length of the dash. For dashed/dotted lines, the length of the interval between dashes equals the length of the dash, and the dot is centered in this interval. For solid lines, the length parameter is ignored.

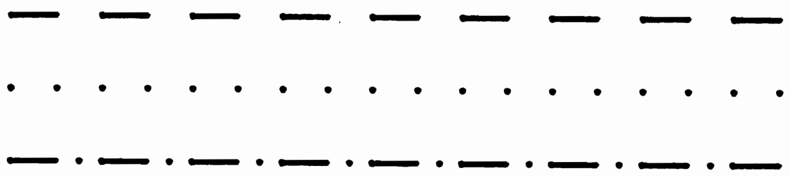


Figure 7-1. Dashed, Dotted, and Dashed/Dotted Lines

The segment length should divide into the line length an odd number of times. If it does not, the subroutine performs a "best fit" routine which automatically adjusts the segment length entered by the user to the next greater length which is divisible into the line length an odd number of times. By thus ensuring that the line length is an odd multiple of the segment length, the program always is able to begin and end the line with a dash (for dashed and dashed/dotted lines) or a dot (for dotted lines).

Option Code

Four Option Codes permit the programmer to specify the type of line to be plotted:

0 = dashed line.
1 = dotted line.
2 = dashed/dotted line.
3 = solid line.

If Option Code 3 (solid line) is specified, the segment length is ignored, and may be set to zero.

7.3 RESTRICTIONS AND EXCEPTIONS

1. If an Option Code other than 3 (solid line) is selected, a segment length of zero causes a Math Error to be signalled at line 1420.

2. If an Option Code other than 3 is selected, a negative value for the segment length produces erratic results, and the desired line is not plotted.

3. If the starting and/or ending point of the line falls outside the active plotting area, the line is clipped at the boundary. Segment and interval lengths are adjusted on the basis of the actual distance between starting and ending points, but only those dashes and dots which fall within the active plotting area are actually plotted.

7.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. `500 DEFFN'25 (M1,M2,M3,M4,L,0)`

2. `150 DEFFN'25 (M1*400, N1*400, M1*400+L*400, N1*400,0,3)`

3. `250 DEFFN'25 (1200,400,2000,6000,20,2)`

CHAPTER 8
PLOT COORDINATE GRID
(DEFFN′ 26)

8.1 PROGRAM DESCRIPTION

The Plot Coordinate Grid Routine plots a grid of horizontal and vertical grid lines within the active plotting area. Horizontal grid lines are plotted from the left-hand boundary of the active plotting area to its right-hand boundary; vertical grid lines are plotted from the lower boundary to the upper boundary. In each case, therefore, the length of the grid lines is determined by the current dimensions of the active plotting area. The intervals between consecutive horizontal grid lines and between consecutive vertical grid lines are specified by the operator, as are the starting points of the first grid lines in each direction.

8.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'26 (E(1), E(2), E(3), E(4))
```

Where:

E(1)	=	Delta X from lower left to first vertical grid line.
E(2)	=	Delta Y from lower left to first horizontal grid line.
E(3)	=	Delta X (interval) between consecutive vertical grid lines.
E(4)	=	Delta Y (interval) between consecutive horizontal grid lines.

NOTES:

1. The variables shown (E(1), E(2), E(3), E(4)) are reserved for use by the subroutine, and should not be used by the programmer in a GOSUB' statement.

2. All distances must be expressed in plotter units.

Delta X, Y To Starting Points of Initial Grid Lines

The programmer must specify the distance along the horizontal axis from the lower left-hand corner of the active plotting area to the starting point of the first vertical grid line, as well as the distance along the vertical axis to the starting point of the first horizontal grid line (see Figure 8-1). The distances are expressed in plotter units (any valid numeric expression may be used).

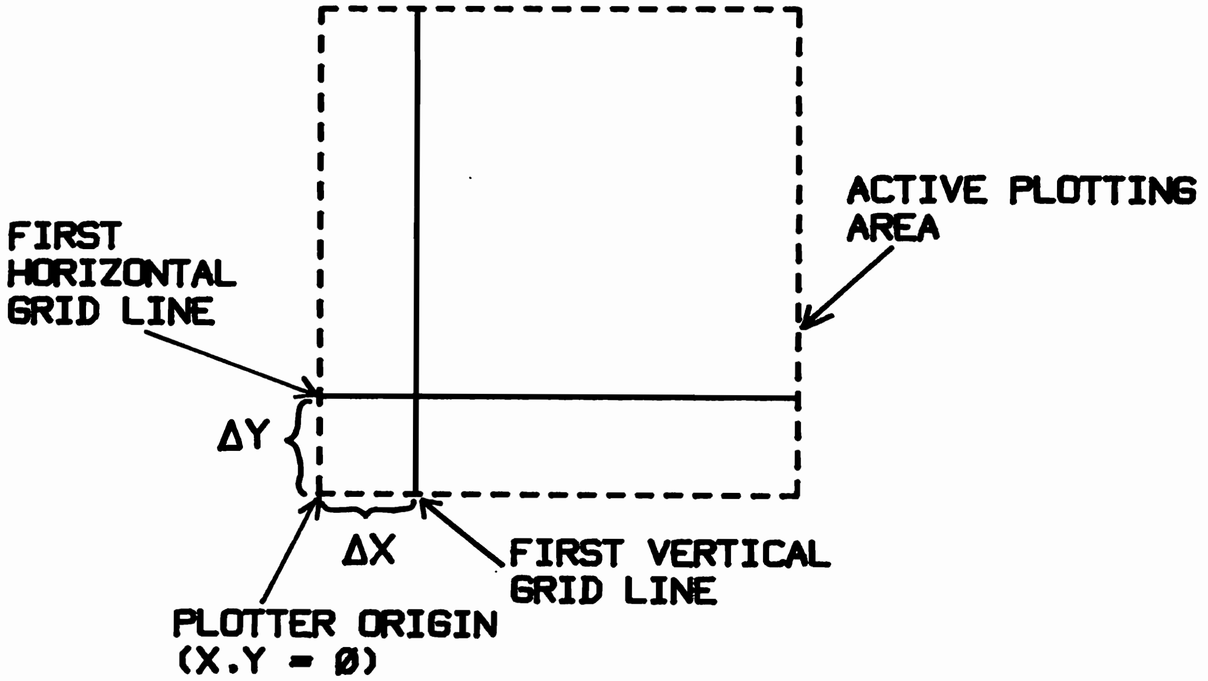


Figure 8-1. Starting Points of Initial Horizontal and Vertical Grid Lines

Delta X, Delta Y Intervals Between Grid Lines

The intervals between consecutive horizontal grid lines, and between consecutive vertical grid lines, must be specified in plotter units (see Figure 8-2).

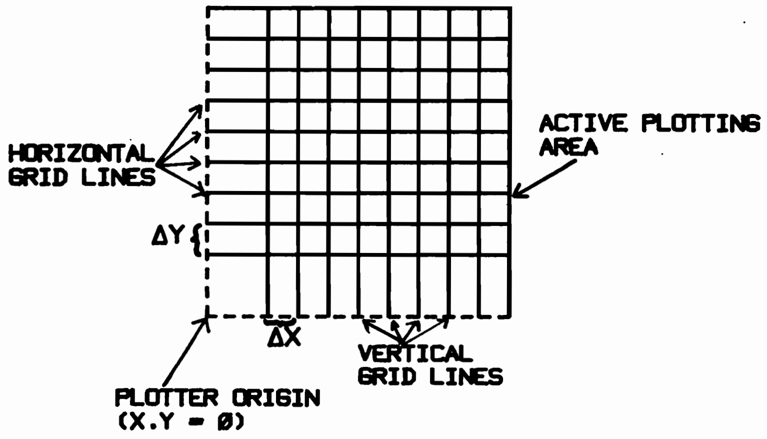


Figure 8-2. Intervals Between Consecutive Grid Lines

8.3 RESTRICTIONS AND EXCEPTIONS

A negative value for any one of the four parameters passed to this routine is illegal, and results in an error.

8.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. `100 GOSUB'26 (400, 400, 400, 400)`
2. `200 GOSUB'26 (M,N,O,P)`
3. `50 GOSUB'26 (M(1)*100,M(2)*100, M(3)*100, M(4)*100)`

CHAPTER 9 PLOT CIRCLE (DEFFN′ 27)

9.1 PROGRAM DESCRIPTION

The Plot Circle routine plots a circle whose center point, radius length, and "degree of smoothness" are specified by the programmer. Through manipulation of the internal angle used to specify the degree of smoothness, the programmer also can produce a variety of regular polygons with this routine.

9.2 SUBROUTINE ARGUMENT LIST

DEFFN'27 (A5, B5, D5, D7)

Where:

A5 = X coordinate of circle center.
B5 = Y coordinate of circle center.
D5 = Radius of circle.
D7 = Central angle enclosing chords used to approximate circle (degrees).

NOTES:

1. The variables shown (A5, B5, D5, D7) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.
2. Coordinates and radius length must be expressed in plotter units.
3. Central angle size must be expressed in degrees.

X,Y Coordinates of Center Point

The coordinates of the circle center point must be specified in plotter units, relative to the plotter origin.

Radius of Circle

The radius length must be specified in plotter units.

Central Angle

Because the plotter cannot draw curved lines, it must approximate a curved line with a series of straight line segments. A circle, therefore, actually is plotted as a regular polygon with a large number of sides. The polygon approaches a circle as the number of sides increases. (See Figure 9-1.)

Each side may be regarded as a chord defined by a central angle of the circle. The number of chords, or sides, is therefore a function of the size of the central angle defining each side. This relationship may be expressed as follows:

$$S = 360/L$$

where 'S' is the number of sides, and 'L' is the central angle (in degrees). Thus a central angle of 10 degrees yields a polygon of 36 sides, an internal angle of 36 degrees yields a polygon of 10 sides, etc.

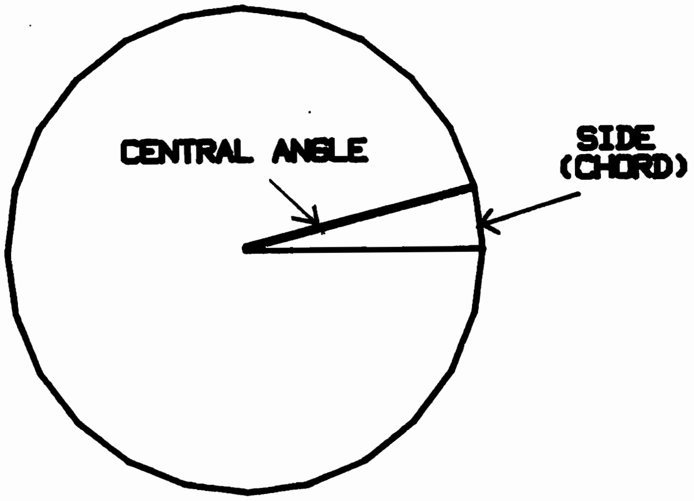


Figure 9-1. Use of Line Segments to Approximate a Circle.

The programmer can control the "smoothness" of the plotted circle by increasing or decreasing the size of the central angle (thus decreasing or increasing the number of sides of the polygon). "Smoothness" is, in this context, a purely subjective characteristic; in general, a circle with a large radius requires a greater number of sides (hence, smaller central angle) to "appear" as smooth as a circle with a smaller radius. It will be necessary for the programmer to experiment with different angles until he achieves the desired effect. If plotting time is a critical factor, it should be noted that the time required to plot a circle increases in proportion to the number of sides.

It should be evident from the preceding discussion that the "Plot Circle" routine is really misnamed. A more accurate name would be "Plot Polygon," since this routine actually permits the user to plot any regular polygon, the number of sides of the polygon being determined by the size of the central angle specified. A central angle of 120^{0}, for example, yields an equilateral triangle; a central angle of 90^{0} yields a square; a central angle of 60^{0} yields a hexagon, etc. (See Figure 9-2.)

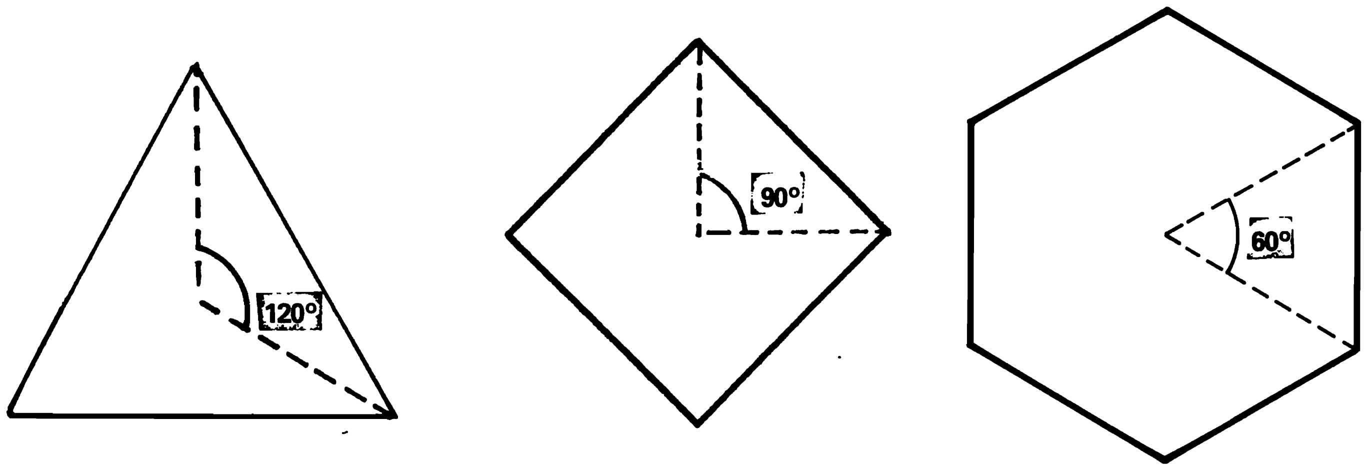


Figure 9-2. Regular Polygons Plotted with the "Plot Circle" Routine

9.3 RESTRICTIONS AND EXCEPTIONS

1. Circle radius must have a length greater than zero.

2. Central angle must have a value greater than zero.

3. The subroutine automatically selects degrees. If the programmer is using radians, he must reselect radians in his main program immediately following subroutine execution.

9.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. `500 GOSUB'27 (4000,2000,600,20)`

2. `200 GOSUB'27 (M1*N,M2*N, L*N, P)`

3. `100 GOSUB'27 (P(1)*400,P(2)*400; P(9)*400,P(6))`

CHAPTER 10
PLOT BORDER AROUND ACTIVE PLOTTING AREA (DEFFN' 28)

10.1 PROGRAM DESCRIPTION

This routine plots a border around the active plotting area (defined with the Set Plotter Boundaries routine (see Chapter 3). The border may consist of a solid line, dashed line, dotted line, or dashed/dotted line.

10.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'28 (C,C1)
```

Where:

C	=	Length of segment/interval.
C1	=	Option Code.
		0 = Dashed.
		1 = Dotted.
		2 = Dashed/Dotted.
		3 = Solid.

NOTES:

1. The variables shown (C,C1) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.

2. If Option Code 3 (Solid Line) is specified, the segment length is ignored, and may be set to zero.

3. For Options Codes 0-2, the segment length must be expressed in plotter units.

Length of Segment/Interval

The length of the line segment (for dashed or dashed/dotted lines) or the interval between dots (for dotted lines) must be specified in plotter units. Refer to Chapter 7, Section 7.1, for a more detailed discussion.

Option Code

Refer to Chapter 7, Section 7.1 for a discussion of the dashed, dotted, dashed/dotted, and solid line options.

10.3 RESTRICTIONS AND EXCEPTIONS

For any one of the Option Codes 0-2, a segment length equal to or less than zero is illegal, and should not be used. With an Option Code of 3 (Solid line), the 'segment length' parameter is ignored, and may be any value.

10.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. `200 GOSUB'28 (15,0)`

2. `500 GOSUB'28 (0,3)`

3. `270 GOSUB'28 (M,1)`

CHAPTER 11
CLEAR SURFACE/PEN SELECT (DEFFN' 24)

11.1 PROGRAM DESCRIPTION

This routine enables the programmer to automatically clear the screen, select PLOT or ERASE modes on the Model 2282 Graphic CRT, and select pens 1, 2, and 3 on the Model 2272. The routine also provides an interrupt and a message for changing surfaces and for changing pens when these operations need to be done manually. Because its application depends on the type of plotter involved, there are three versions of this routine:

Model 2282 Graphic CRT
1 pen plotters (Models 2212, 2232B)
3 pen plotter (Model 2272-2)

NOTE:

The version required is selected by the user during the loading routine.

11.2 SUBROUTINE ARGUMENT LIST (FOR MODEL 2282 GRAPHIC CRT)

DEFFN'24

Where:

D8 = Option Code.
  0 = Clear Screen.
  1 = Pen Select.

D9 = Option Code.
  -1 = Erase.
  X = Select Pen 1 (for all X $\neq$ -1).

NOTES:

1. The variables shown (D8, D9) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.

2. If the Option Code 0 (Clear Screen) is specified, Option (D9) is ignored and may be set to zero.

Clear Screen

The Graphic CRT screen is cleared of all plotted vectors.

Pen Select

Indicates that the Select Pen routines will be used

Erase Mode

All illuminated dots on a specified vector will extinguish.

Pen 1 (Plot Mode)

Dots corresponding to the specified line or point will light up on the Graphic CRT screen. (Note: This is the default mode and is automatically used unless changed to Erase Mode.)

11.3 SUBROUTINE ARGUMENT LIST (FOR 1 PEN PLOTTER)

DEFFN'24 (D8, D9)

Where:

D8 = Option Code.
0 = New Surface.
1 = Pen Select.

D9 = Option Code.
0 = Initialize.
X = Select Pen 1 (X is an integer from 1 to 99).

NOTES:

1. The variables shown (D8, D9) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.

2. The subroutine uses an internal table for recording the status of the pen on the plotter (pen mounted or unmounted).

<u>New Surface</u>

The routine will interrupt the program, display a prompt message to change the surface, and wait until the operator keys RETURN(EXEC).

<u>Pen Select</u>

Indicates that the Select Pen routines will be used.

<u>Initialize</u>

This indicates on the internal table that pen 1 is mounted. It is the responsibility of the operator to ensure that pen 1 is actually on the plotter. After pen changes have been made, the internal table can be reinitialized by selecting the second Option Code = 0. (Note: The table is automatically initialized the first time GOSUB'24 is called. It is not necessary to initialize it separately.)

<u>Pen 1</u>

The routine will examine the pen specified in the second Option Code (D9) and if the pen is not recorded on the internal table as already being mounted, it will interrupt the program, display a prompt message to mount the pen, and wait until the operator keys RETURN(EXEC). Note: If the desired pen is indicated by the internal table as being mounted, no pause will occur. It is important that the operator ensure that the mounted pen is the same as the pen indicated by the internal table. Both the pen indicated by the internal table and the pen which is to be mounted are displayed by the prompt.

11.4 SUBROUTINE ARGUMENT LIST (FOR 3 PEN PLOTTER)

DEFFN'24 (D8, D9)

Where:

D8 = Option Code.
- 0 = New Surface.
- 1 = Pen Select.

D9 = Option Code.

0 = Initialize.		
1 =	Select Pen 1	Position 1.
2 =	Select Pen 2	Position 2.
3 =	Select pen 3	Position 3.
X =	Select Pen X	Position 1 (X is an integer from 4 to 99).

NOTES:

1. The variables shown (D8, D9) are reserved for use by the subroutines and should not be used by the programmer in a GOSUB' statement.

2. The subroutine uses an internal table for recording the status of each pen on the plotter (pen mounted or unmounted.)

New Surface

Same as for 1 Pen Plotters.

Pen Select

Same as for 1 Pen Plotters.

Initialize

This indicates on the internal table that pen 1 is mounted in position 1, pen 2 is mounted in position 2, and pen 3 is mounted in position 3. It is the responsibility of the operator to ensure that the correct pens are actually mounted in the correct positions on the plotter. After pen changes have been made, the internal table can be reinitialized by selecting the second Option Code = 0. (Note: The table is automatically initialized the first time GOSUB'24 is called. It is not necessary to initialize it separately.)

Pen 2 and Pen 3

The routine will automatically select Pen 2 and Pen 3 as required by the program with no pause.

Pen 1 and Pen X

The routine will examine the pen specified in the second Option Code, and if it is not recorded on the internal table as already being mounted, it will interrupt the program, display a prompt message to mount the pen and wait until the operator keys RETURN(EXEC). If the desired pen is indicated by the internal table as already being mounted, no pause will occur. The operator must ensure that the mounted pens are the same as the pen indicated by the internal table. both the pens indicated as being mounted (and their positions) by the internal table and the pen which is to be mounted are displayed by the prompt. Pens are to be changed in position 1 only since the internal table always indicates pen 2 and pen 3 in their respective positions.

11.5 RESTRICTIONS AND EXCEPTIONS

1. For the Graphic CRT, if the first Option Code ≠0, Pen SELECT will result. If the second Option Code ≠-1, the Plot Mode will be selected.

2. For the one and three pen plotters, if the first Option Code ≠0, PEN SELECT will result. If negative or excessively large or non-integral values are specified for the second Option Code, errors will result or misleading values may be stored in the internal table.

11.6 EXAMPLES OF VALID GOSUB' SYNTAX

1.	100 GOSUB'24 (0, 0)	
2.	200 GOSUB'24 (1, -1)	For Graphic CRT.
3.	300 GOSUB'24 (1, 0)	To initialize 1 and 3 pen plotters.
4.	400 GOSUB'24 (1, 1)	
5.	500 GOSUB'24 (1, 3)	Requires pause on 1 pen plotter.
6.	600 GOSUB'24 (1, 6)	Requires pause on 1 and 3 pen plotters.

CHAPTER 12
PLOT INSTRUCTION EMULATOR
(DEFFN′ 29)

12.1 PROGRAM DESCRIPTION

The Plot Instruction Emulator enables the programmer to perform a variety of simple plotter operations with a single, uncomplex routine. (All the special plotter routines described in Chapters 3-11 call the Instruction Emulator to execute plotting operations.) Its versatility and simplicity,combine to make the Plot Instruction Emulator a powerful programming tool for general plotter control.

Three parameters are passed to the Plot Instruction Emulator routine: an X value, a Y value, and an Instruction Code. The type of operation to be carried out is determined by the specified Instruction Code (nine such codes are available). The Instruction Codes permit three types of plotter operations: plot point, plot line, and move plotter with pen up (no plotting). In each of these three cases, the routine offers the programmer the choice of working in absolute plotter coordinates (relative to the plotter origin), or in delta values (relative to the current plotter position). Of the nine Instruction Codes, all even-numbered instructions (2,4,6,8) operate with absolute coordinates, and all odd-numbered instructions (1,3,5,7) operate with delta values. Instruction Code 0 simply sends the plotter home.

In addition to the alternatives of absolute plotter coordinates or delta values, the Instruction Emulator also permits two types of plotter movement: "actual" moves and "tacit" moves. an actual move causes the plotter to physically move to the specified coordinates (or the specified X and Y distances from the current position), while a tacit move causes the routine to adjust its internal pointers to the new specified plotter position, without physically moving the plotter. Tacit moves can save a great deal of plotting time in situations where a lot of plotter movement precedes the plotting of a point.

12.2 SUBROUTINE ARGUMENT LIST

DEFFN'29 (A,B,D)

Where:

A = X Value (X coordinate or delta X).
B = Y Value (Y coordinate or delta Y).
D = Instruction Code.
- 0 = Send plotter home (parameters A and B are ignored).
- 1 = Move delta X, delta Y (Tacit Move).
- 2 = Move to X;Y (Tacit Move).
- 3 = Move delta X, delta Y (Actual Move).
- 4 = Move to X,Y (Actual Move).
- 5 = Move delta X, delta Y, plot point.
- 6 = Plot point at X,Y.
- 7 = Plot line delta X, delta Y.
- 8 = Plot line to X,Y.

NOTES:

1. The variables shown (A,B,D) are reserved for use by the subroutine and should not be used by the programmer in a GOSUB' statement.
2. Coordinates must be specified in plotter units, relative to the plotter origin.
3. Delta values must be expressed in plotter units.

X,Y Coordinates vs. Delta X, Delta Y

Of the nine Instruction Codes provided by the Plot Instruction Emulator, four utilize delta values (1,3,5,7) and four utilize plotter coordinates (2,4,6,8). The Instruction Code selected determines whether the first two parameters are to be interpreted as X,Y coordinates or as distances along the X and Y axes from the current effective plotter position (i.e., as X,Y delta values). In either case, the values must be expressed in plotter units. X and Y coordinates always are relative to the plotter origin.

Instruction Code

The nine Instruction Codes are:

0 - Send plotter home (first two parameters ignored).
1 - Move delta X, delta Y (Tacit Move).
2 - Move to X,Y (Tacit Move).
3 - Move delta X, delta Y (Actual Move).
4 - Move to X,Y (Actual Move).
5 - Move delta X, delta Y, plot point.
6 - Plot point at X,Y.
7 - Plot line delta X, delta Y.
8 - Plot line to X,Y.

An Instruction Code of 0 causes the plotter to return to the home position (plotter origin), and remain there with the pen up. In this case, the first two arguments passed are ignored and may be set to zero.

Instruction Code 1 causes a tacit move with delta values. The X value is interpreted as a delta X and is added to the current X coordinate; the Y value is interpreted as a delta Y and is added to the current Y coordinate. The updated coordinates represent the new effective plotter position, but the plotter is not physically moved to this location. Any subsequent delta instruction functions with reference to the effective plotter position, not the physical plotter position.

Instruction Code 2 also produces a tacit move, this time with absolute coordinates. The X and Y values are interpreted as absolute X and Y coordinates (relative to the plotter origin), and these coordinates are substituted for the current X and Y coordinates to identify the new effective plotter position. The plotter is not physically moved to the new location, however. Any subsequent delta instruction functions with reference to the current effective plotter position, not the current physical plotter position.

Instruction Code 3 produces an actual move with delta values. The X and Y values are interpreted as delta values and are added respectively to the current X and Y coordinates. The plotter is then physically moved to the new position with the pen up (no plotting).

Instruction Code 4 produces an actual move with absolute coordinates. The X and Y values are interpreted as X and Y coordinates, and the plotter is physically moved to the new position with the pen up (no plotting).

Instruction Code 5 causes the plotter to move a specified distance from the current effective plotter position and plot a point. The X and Y values are interpreted as delta values and are added to the X and Y coordinates of the current effective plotter position to yield the coordinates at which the point will be plotted. The plotter moves to this position with the pen up, plots a point, and remains with pen up.

Instruction Code 6 causes the plotter to move to a specified position and plot a point. The X and Y values are interpreted as absolute plotter coordinates defining the location at which the point will be plotted. The plotter moves to this point with pen up, plots a point, and remains with the pen up.

Instruction Code 7 causes the plotter to plot a line beginning at the current effective plotter position, and ending at a point which is a specified distance from the current plotter position. The X and Y values are interpreted as delta values, and are added to the X and Y coordinates of the current plotter position to yield the coordinates of the end point. The plotter then draws a line from the current effective plotter position to the end point, and remains at the end point with pen down.

Instruction Code 8 causes the plotter to plot a line beginning at the current effective plotter position, and ending at a specified point. The X and Y values are interpreted as absolute plotter coordinates of the ending point of the line. The plotter draws a line from the current effective plotter position to the end point, and remains at the end point with pen down.

12.3 RESTRICTIONS AND EXCEPTIONS

1. Absolute coordinates must be positive values (delta values may either be positive or negative).

2. The plotter should not be instructed to move or plot outside the physical plotter boundaries.

12.4 EXAMPLES OF VALID GOSUB' SYNTAX

1. 50 GOSUB'29 (0,0,0)

2. 500 GOSUB'29 (M1,M2,M3)

3. 100 GOSUB'29 (M*100, N*100, P)

CHAPTER 13
PLOTTER CONTROL ROUTINE (DEFFN'30)

13.1 PROGRAM DESCRIPTION

The Plotter Control Routine is the subroutine which directly controls plotter movement. It is called by the Plot Instruction Emulator in order to execute plotting instructions. Because it interacts directly with the plotter hardware, the Plotter Control Routine is the only device-dependent routine in the Plotter Utilities Package. Five versions of the routine are supplied: one version controls the Model 2202, a second version controls the Models 2212 and 2232B, a third version controls the Model 2272 and the Model 2282, a fourth version controls the Tektronix Graphic Display Terminal, and a fifth version controls the Model 2281P.

The Plotter Control Routine is designed to be called from the Plot Instruction Emulator and should never be accessed directly by the programmer.

13.2 SUBROUTINE ARGUMENT LIST

```
DEFFN'30 (A,B,D)
```

Where:

```
A = X coordinate.
B = Y coordinate.
D = Option Code.
    0 = Pen down.
    1 = Pen up.
    2 = Point.
    3 = Home.
```

13.3 RESTRICTIONS AND EXCEPTIONS

None. This routine should not be called directly from the main line program. It is designed to be accessed only via the Plot Instruction Emulator routine.

APPENDIX A: GENERAL FORMS OF THE PLOTTER UTILITY ROUTINES

NOTE:

All variables shown in the subroutine general forms are those actually used by the subroutine and should not be used by the programmer in a GOSUB' statement which calls the subroutine. (In general, reserved variables should not be used at all in the user's application program; refer to Appendix D for a complete list of reserved variables.)

A.1 SET PLOTTER BOUNDARIES

```
DEFFN'19 (A8,B8,A9,B9,D4)
```

Where:

```
A8 = X coordinate of lower left corner.
88 = Y coordinate of lower left corner.
A9 = X coordinate of upper right corner.
B9 = Y coordinate of upper right corner.
D4 = Option Code.
     0 - Reset boundaries (no plotter movement).
     1 - Initialize boundaries (move to lower left).
     2 - Reset boundaries (move to lower left).
```

A.2 PLOT CHARACTER STRING (STRAIGHT LINE)

```
DEFFN'20 (C$,C,C6,C7,C1,C2,C3)
```

Where:

```
C$ = Character String.
C  = Character size.
C6 = Character slant reference angle (degrees).
C7 = Character rotation (degrees).
C1 = X coordinate of reference point.
C2 = Y coordinate of reference point.
C3 = Option Code.
     0 - Start at X,Y.
     1 - Center at X,Y.
     2 - End at X,Y.
```

A.3 PLOT CHARACTER STRING (ON A CIRCLE)

DEFFN'21 (E$,E1,E2,E3,E4,E5,E6,E7)

Where:

E$ = Character String.
E1 = Character size.
E2 = Character slant reference angle (degrees).
E3 = X coordinate of circle center.
E4 = Y coordinate of circle center.
E5 = Radius length.
E6 = Reference point on circle (degrees).
E7 = Option Code.
- 0 - Start at reference point.
- 1 - Center at reference point.
- 2 - End at reference point.

A.4 LOAD CHARACTER GENERATION ARRAY

DEFFN'22 (C$,C)

Where:

C$ = Name of character data file.
C = File number assigned to disk address.

A.5 GIN MODE ROUTINE

DEFFN'23

The GIN Mode Routine is used to receive graphic input from models of the Tektronix graphic terminal which support a GIN mode capability. The subroutine has no arguments.

A.6 PLOT LINE BETWEEN TWO POINTS

DEFFN'25 (F,F0,F1,F2,F3,F4)

Where:

F = X coordinate of starting point.
F0 = Y coordinate of starting point.
F1 = X coordinate of ending point.
F2 = Y coordinate of ending point.
F3 = Length of segment or interval.
F4 = Option Code.
- 0 - Dashed line.
- 1 - Dotted line.
- 2 - Dashed/dotted line.
- 3 - Solid line (segment length ignored).

A.7 PLOT COORDINATE GRID

```
DEFFN'26 (E(1), E(2), E(3), E(4))
```

Where:

```
E(1) = Delta X from lower left to first vertical grid line.
E(2) = Delta Y from lower left to first horizontal grid line.
E(3) = Delta X between vertical grid lines.
E(4) = Delta Y between horizontal grid lines.
```

A.8 PLOT CIRCLE

```
DEFFN'27 (A5,B5,D5,D7)
```

Where:

```
A5 = X coordinate of circle center.
B5 = Y coordinate of circle center.
D5 = Length of radius.
D7 = Central angle (defines length of line segments used in
     approximating circle).
```

A.9 PLOT BORDER AROUND ACTIVE PLOTTING AREA

```
DEFFN'28 (C,C1)
```

Where:

```
C  = Length of segment or interval.
C1 = Option Code.
     0 - Dashed line.
     1 - Dotted line.
     2 - Dashed/dotted line.
     3 - Solid line (segment length ignored).
```

A.10 CLEAR SURFACE/PEN SELECT

```
DEFFN'24 (D8,D9)
```

Where:

```
D8 = Option Code
    0 - New Surface/Clear Screen
    1 - Pen Select

D9 = Option Code.

   -1 - Erase (Model 2282 only).
    0 - Initialize (Models 2212, 2232B, 2272).
    1 - Select Pen 1, Position 1 (Model 2272).
    2 - Select Pen 2, Position 2 (Model 2272).
    3 - Select Pen 3, Position 3 (Model 2272).
    X - Select Pen 1 (Plot Mode, Model 2282).
    X - Select Pen 1 (Models 2212, 2232B).
    X - Select Pen X (Model 2272)
```

A.11 PLOT INSTRUCTION EMULATOR

DEFFN'29 (A,B,D)

Where:

A = X coordinate or delta X.
B = Y coordinate or delta Y.
D = Instruction Code.
0 - Send plotter home (first two parameters ignored).
1 - Move delta X, delta Y (Tacit Move).
2 - Move to X,Y (Tacit Move).
3 - Move delta X, delta Y (Actual Move).
4 - Move to X,Y (Actual Move).
5 - Plot point delta X, delta Y.
6 - Plot point at X,Y.
7 - Plot line delta X, delta Y.
8 - Plot line to X,Y.

A.12 PLOTTER CONTROL ROUTINE

DEFFN'30 (A,B,D)

Where:

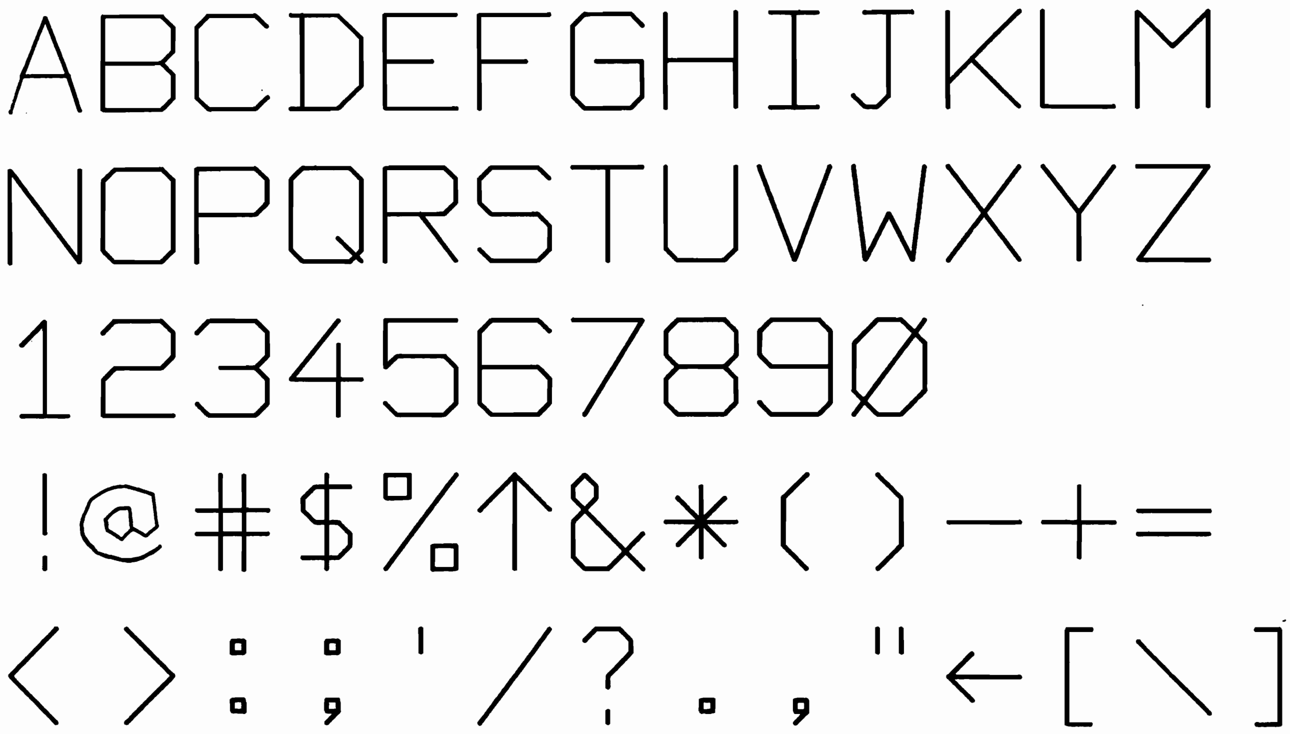
A = X coordinate.
B = Y coordinate.
D = Option Code.
0 - Pen up.
1 - Pen down.
2 - Plot point.
3 - Send plotter home.

NOTE:

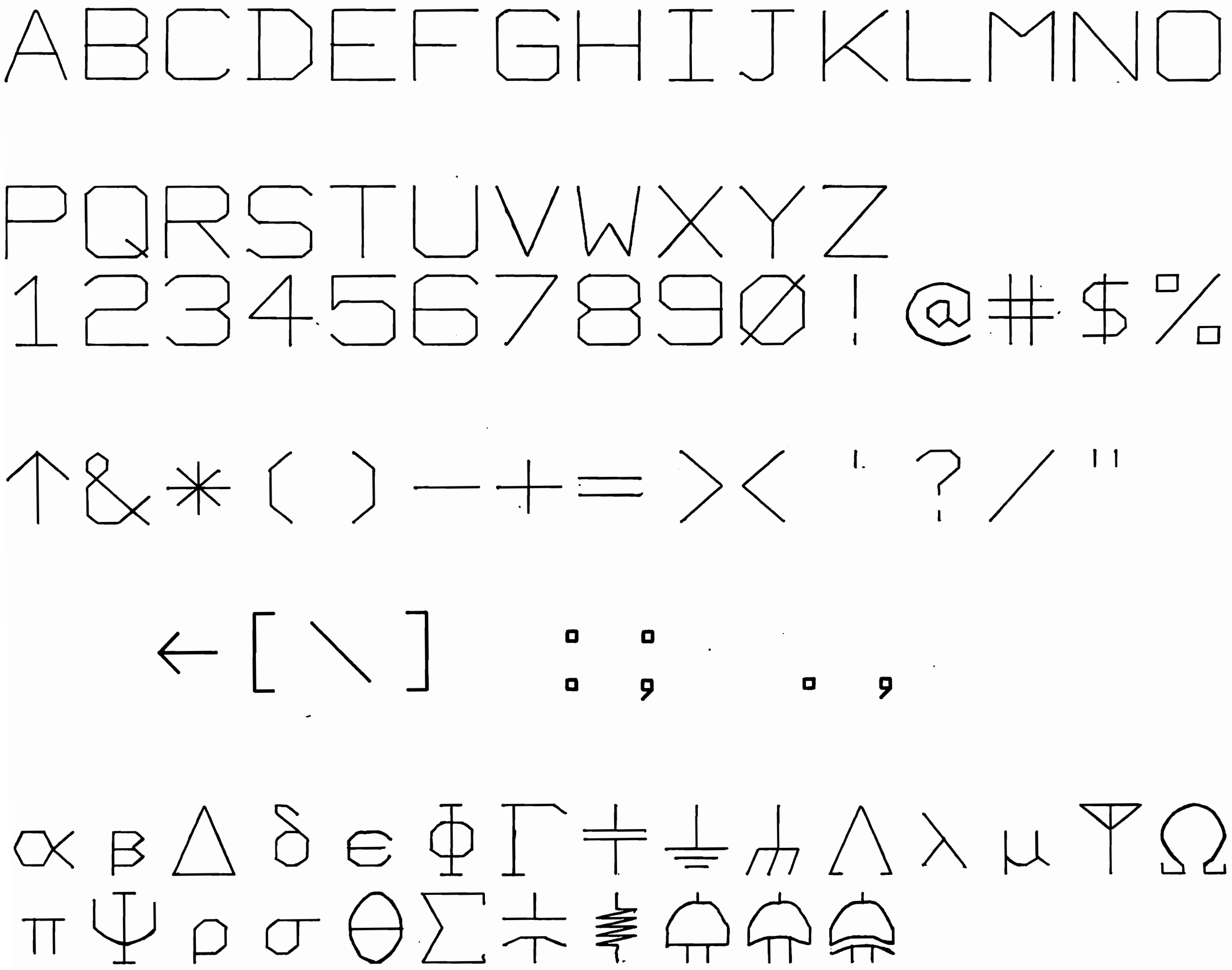
The Plotter Control Routine is designed to be accessed via the Plot Instruction Emulator (DEFFN'29). This routine should not be accessed directly from the user's software.

APPENDIX B: CHARACTER SETS

STANDARD CHARACTER SET FOR ALL PLOTTERS



ENHANCED CHARACTER SET FOR ALL PLOTTERS



APPENDIX C: CUSTOMIZING THE PLOTTER CHARACTER SET

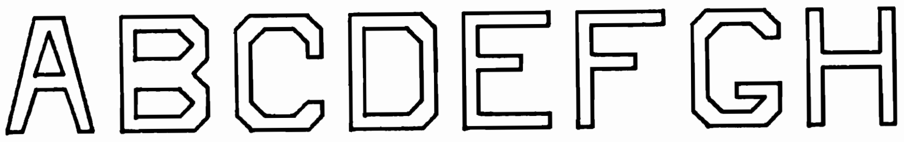


Figure C-1. Some Special Characters Created by Customizing the Character Generation Array.

C.1 INTRODUCTION

If the programmer has labelling applications which require special characters not included in either the Standard Character Set, or the Enhanced Character Set, he must define these characters himself. Each character is defined with a sequence of coded plotting instructions, which must be stored in the character generation array. The remainder of this appendix describes the procedure for creating and storing such code sequences.

C.2 THE CHARACTER GENERATION ARRAY

The character generation array is actually created in two steps: initially, the coded plotting sequences for all characters are stored in array A$(); subsequently, the contents of A$() are packed into a second array, C$(). It is the packed array, C$(), which is stored on disk, and is recalled into memory with the Load Character Array Routine. The packing operation permits more efficient use of memory and faster search time within the array. The packing operation is carried out automatically by the PLOT001A (or PLOT002A) subroutine; once the programmer has stored the necessary code sequences in A$(), the subroutine will take care of packing this information into C$().

NOTE:

The PLOT001A subroutine is used with the Standard Character Set; the PLOT002A subroutine is used with the Enhanced Character Set.

Figure C-2 is a listing of lines 10-260 of the PLOT001A program, which is called by the START module to initialize the character generation array when such initialization is requested by the operator (refer to Chapter 2). From line 20 of this listing, it can be seen that A$() is a one-dimensional array containing 63 elements, each 17 bytes in length. Each element contains the coded plotting sequence for one of the characters in the standard character set. (Note that the 64th character in the standard character set is the space, which is not defined with a unique plotting sequence, and therefore need not be stored in the character array.)

```
10 REM PLOT001A,02-00(12/12/77),3 23 2 00 - COPYRIGHT WANG LABS. INC., 1977
20      DIM A$(63)24,C$(5)61,C$1
30 REM SET UPPER LIMIT OF A$()
40      A = 63
50 REM ----------------------------------------------
60 REM DEFINE CHARACTER SET
70 REM !
80      A$(1) = HEX(817C194B04)
90 REM "
100      A$(2) = HEX(827B2D7D2F)
110 REM #
120      A$(3) = HEX(837B03453D6A24561C)
130 REM $
140      A$(4) = HEX(847C04490C141B211F252C3437)
150 REM %
160      A$(5) = HEX(857B2D2B393B7F014513150705)
170 REM &
180      A$(6) = HEX(86472B323A342D1D0802041C)
190 REM '
200      A$(7) = HEX(877C2E)
210 REM (
220      A$(8) = HEX(887D2D1105)
230 REM )
240      A$(9) = HEX(897B2F1303)
250 REM *
260      A$(10) = HEX(8A5D2370104B356C14)
```

Figure C-2. Partial Listing of PLOT001A, Showing Assignments of Coded Plotting Sequences for Individual Characters to Elements of A$().

The coded plotting instructions for each character are represented as a unique sequence of hexadecimal codes stored in an element of A$(). In the PLOT001A listing (Figure C-2), observe that a REM statement immediately above the assignment statement for each element of A$() shows the character plotted by the code sequence stored in that element. A$(1), for example, contains the coded sequence for plotting an exclamation point (!).

NOTE:

The line numbers for PLOT001A shown in Figure C-2 are those of the most recent version of the program available at the printing of this manual. Since subsequent updates may alter the line numbers, you should not assume that the line numbers shown in Figure C-2 are identical to those in your version of the PLOT001A routine.

Every coded sequence is composed of the following elements:

1. A character I.D. code, which always occupies the first byte (i.e., the first pair of hexadecimal digits) in the sequence.

2. A series of coded plot instructions, which occupy a varying number of bytes following the control code, up to a maximum of 23. (Each plot instruction consists of a pair of hexadecimal digits, and occupies one byte.)

For example, the code sequence for plotting an exclamation point is divided into two components as shown in Figure C-3:

$$\underbrace{81}_{\text{Character I.D. Code}}\underbrace{7C194B04}_{\text{Coded Plotting Instructions}}$$

Figure C-3. Code Sequence for Plotting Exclamation Point

The character I.D. code, which is used to identify each plotting sequence, is discussed in Section C.4. The following section (C.3) explains the procedure for coding plot instructions.

C.3 CODING PLOT INSTRUCTIONS FOR CHARACTER PLOTTING

It should be recalled from Chapter 4 that every character is defined within a 9 x 11 matrix of vertices. This 9 x 11 matrix includes a surrounding buffer which is not part of the character itself; however, the character is actually defined in a 7 x 9 matrix embedded within the larger matrix. Because the buffer is created automatically by the subroutine, it is not of concern to the programmer, who needs to direct his attention exclusively to the 7 x 9 matrix used for character definition.

Each 7 x 9 matrix consists of a total of 63 discrete "vertices," or defined points. The plotter can be instructed to move to any one of the 63 vertices with pen up, or to plot a line between any pair of vertices. Figure C-4 shows the 7 x 9 matrix of vertices.

```
39/79  3A/7A  3B/7B  3C/7C  3D/7D  3E/7E  3F/7F
  *      *      *      *      *      *      *

32/72  33/73  34/74  35/75  36/76  37/77  38/78
  *      *      *      *      *      *      *

2B/6B  2C/6C  2D/6D  2E/6E  2F/6F  30/70  31/71
  *      *      *      *      *      *      *

24/64  25/65  26/66  27/67  28/68  29/69  2A/6A
  *      *      *      *      *      *      *

1D/5D  1E/5E  1F/5F  20/60  21/61  22/62  23/63
  *      *      *      *      *      *      *

16/56  17/57  18/58  19/59  1A/5A  1B/5B  1C/5C
  *      *      *      *      *      *      *

0F/4F  10/50  11/51  12/52  13/53  14/54  15/55
  *      *      *      *      *      *      *

08/48  09/49  0A/4A  0B/4B  0C/4C  0D/4D  0E/4E
  *      *      *      *      *      *      *

01/41  02/42  03/43  04/44  05/45  06/46  07/47
  *      *      *      *      *      *      *
```

Figure C-4. The 7 x 9 Matrix of Vertices Used to Define a Character (Vertices are Represented by Asterisks, *)

Note that each vertex is identified by a pair of 2-digit hexadecimal numbers. The numbers on the left-hand side of the slashes run sequentially from 01 (lower left corner) to 3F (upper right corner). You should recognize that 3F is the hexadecimal equivalent of decimal 63. The vertices are therefore numbered sequentially in hexadecimal from 1 to 63.

The hexadecimal numbers on the right-hand side of the slashes run sequentially from 41 (lower left) to 7F (upper right). Careful examination of each pair of hex numbers will disclose that the right-hand number in each pair is always HEX(40) greater than the left-hand number. For example, HEX(41) is greater by HEX(40) than HEX(01), and HEX(7F) is greater by HEX(40) than HEX(3F). It should be further noted that in the left-hand set of numbers 01-3F, the 4-bit in the high-order hex digit is always set to 0. Correspondingly, in the right-hand set of numbers 41-7F, the 4-bit is always set to 1. The significance of this distinction is as follows:

1. If the high-order 4-bit is OFF (i.e. = 0), the plotter is instructed to plot a line to the specified vertex.

2. If the high-order 4-bit is ON (i.e., = 1), the plotter is instructed to move to the specified vertex with pen up (i.e., no plotting).

It should be clear, then, that the left-hand set of hex numbers represent coded "plot" instructions, while the right-hand set of hex numbers represent coded "move" instructions. For example, the code <u>41</u> says, in effect, "move to vertex <u>41</u>," while the code 3F says "plot a line from current position to vertex 3F." The sequence of codes 413F therefore has the effect of plotting a line diagonally from the lower left corner of the matrix to the upper right corner.

An example may be helpful in illustrating the use of these coded instructions. A$(3) contains the coded instruction sequence for the character "#". The assignment statement on line 120 of the INITALPH program shows that "#" is defined with the following sequence:

```
120 A$(3) = HEX (837B03453D6A24561C)
```

The first code, 83, is disregarded (since it is the character I.D. code, and not part of the plotting sequence). The plotting sequence can then be decoded in the following way:

```
7B = Move to 7B
03 = Plot to 03
45 = Move to 45
3D = Plot to 3D
6A = Move to 6A
24 = Plot to 24
56 = Move to 56
1C = Plot to 1C
```

Figure C-5. Decoded Plotting Sequence for Character '#'

If the plotting sequence in Figure C-5 is compared with the '#' character sketched onto the 9 x 7 matrix in Figure C-6, it will be seen that the plot defined by these instructions produces the special character '#'.

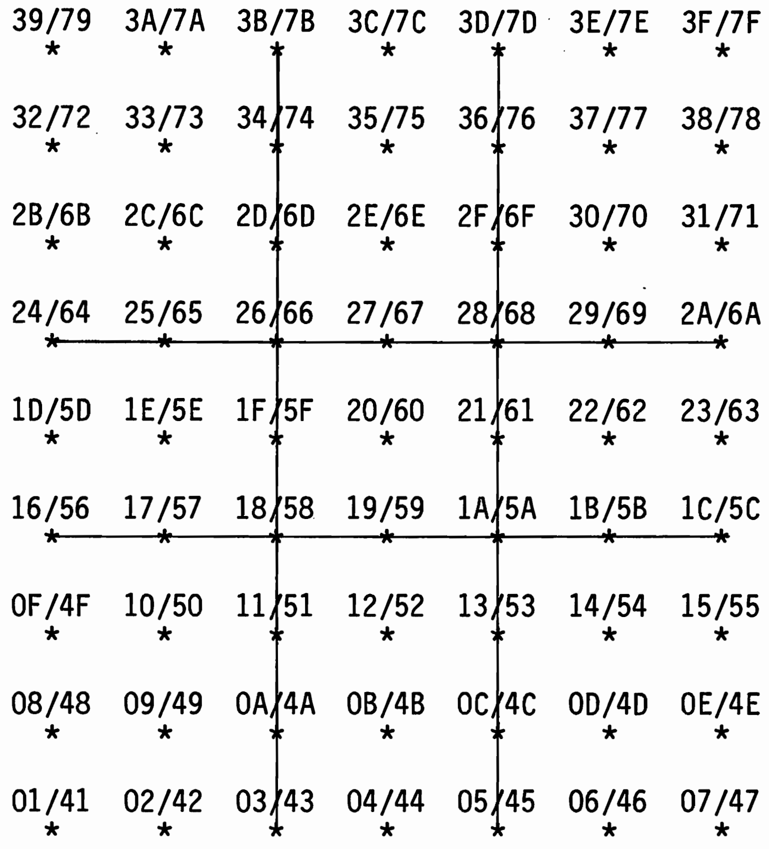


Figure C-6. "#" Character Sketched on 7 x 9 Matrix of Vertices

The procedure for defining a character thus turns out to be astonishingly simple. First, sketch the character on a 7 x 9 matrix identical to the one in Figure C-4. Experiment until a character with the most pleasing appearance is produced. (The recommended procedure is to place a clear plastic sheet over the matrix in Figure C-4, and draw the character on the plastic with a grease pencil or similar implement. The plastic can be wiped clean, and the character redrawn, several times, until the right character is produced.) Next, translate the track followed by your pencil when tracing the character into a sequence of coded instructions which direct the plotter to follow the same path. The plotted result will (if you are careful) be identical to your sketch.

C.4 THE CHARACTER I.D. CODE

Once a plotting sequence for a particular character has been defined, it must be identified in some way. Suppose, for example, that the following subroutine call were executed:

```
    200 GOSUB'20 ("#", 10, 0, 0, M, N, 0)
```

This GOSUB' statement calls the Plot Character String (Straight Line) Routine to plot a '#' character. In the preceding section, you saw how this special character is defined with a coded plotting sequence stored in A$(3). The question now arises: how does the subroutine connect a '#' character entered from the keyboard with the plotting sequence stored in A$(3)? The answer is: by means of the character I.D. code.

The System 2200 utilizes the ASCII coding scheme, an industry standard coding technique, for representing alphanumeric characters internally. Each character in the ASCII character set is assigned a unique two-digit hexadecimal number as its code. The ASCII code for the special character '#', for example, is HEX(23). (A complete list of ASCII codes for alphanumeric characters can be found in Appendix A of your Wang BASIC Language Reference Manual.) Whenever a key representing an alphanumeric character is depressed, the ASCII code for that character automatically is transmitted to the system. Thus, depressing the '#' key automatically sends a HEX(23) code to the CPU. It may be seen, then, that the following pair of GOSUB' statements have identical results:

```
    100 GOSUB'20 ("#", 10, 0, 0, M, N, 0)
or
    90 A$ = HEX(23)
    100 GOSUB'20 (A$, 10, 0, 0, M, N, 0)
```

It would be convenient if the ASCII code for each character could be used directly as its character I.D. code in the Character Generation Array. Unfortunately, this is not possible, because many of the hexadecimal codes used as ASCII character codes are utilized in the Character Array as coded plotting instructions. (For example, HEX(23) is a coded instruction which means "plot to vertex 23".) The subroutine would therefore have no way of distinguishing between a hex code which represents a character I.D. code and a hex code which represents a plot instruction.

This ambiguity is avoided by logically ADDing HEX(60) to each ASCII code. When performed on ASCII codes greater than HEX(19), the ADD operation always has the effect of setting the high-order 8-bit to 1. (For example, if HEX(60) is ADDed to HEX(23), the result, HEX(83), has a 1 in the 8-bit position of the high-order (left-hand) digit.) Since the high-order 8-bit is always set to 0 in the coded plot instructions, the presence of a 1 in the high-order 8-bit position enables the subroutine to distinguish unambiguously between a hexadecimal code which is used as a character I.D. code and a hexadecimal code which is to be interpreted as a coded plotting instruction. Recall, for example, the coded sequence for the special character '#':

```
A$(3) = HEX(837B03453D6A24561C)
            ↑
        Character I.D. Code
```

Note that the character I.D. code, HEX(83), is the result of ADDing HEX(60) to HEX(23), the ASCII code of the '#' character.

When a character string is passed to one of the plot character subroutines, the subroutine ADDs HEX(60) to the ASCII code of each character, and searches for the resultant character I.D. code in the character generation array. The coded plotting instructions which follow the character I.D. code are then read and executed.

When a new character is defined, the programmer must identify it with a unique character I.D. code. If the newly-defined character is a keyboard character, its ASCII code can be used to create the character I.D. code. In the event the new characters are special mathematical symbols, or other characters not found on the keyboard, there are several ways of creating new character I.D. codes for them.

NOTE:

The uppercase characters, numbers and symbols of the keyboard are used to define most of the Standard Character Array. The Greek letters and electronic logic symbols of the Enhanced Character Array are defined by the lowercase letters a through z of the keyboard. (See Appendix H).

If the new character or characters are to replace existing characters in the system-generated character array, the I.D. codes of the existing characters can be used for the new characters. If, for example, the exclamation point is not wanted, its location (A$(1)) could be used to store the coded plotting sequence for a new character, and its character I.D. code (81) assigned to the new character. In this case, the exclamation point key (!) identifies the new character, and the new character will be plotted whenever the exclamation point is passed to a plot character subroutine.

Alternatively, the programmer may prefer to utilize the codes of those keyboard characters which are not defined in the character generation array.

Finally, the programmer may elect to use hexadecimal codes other than those used in the ASCII character set. In this case, no keyboard character can be used to generate the initial code; it must be specified in a HEX function. The HEX function is stored in an alphanumeric variable, and the variable is specified in the subroutine argument list. For example:

```
90  A$ = HEX(5F)
100 GOSUB'20 (A$, 10, 0, 0, M, N, 0)
```

Note that there are restrictions on the range of hexadecimal codes which can be used in the creation of a character I.D. code (see the list of restrictions in the final section of this appendix). Remember, finally, that the hex code passed to the subroutine is not the character I.D. code; the subroutine will automatically ADD a HEX(60) to the ASCII code it receives in order to produce the character I.D. code.

C.5 <u>INSERTING NEW CHARACTERS IN THE CHARACTER GENERATION ARRAY</u>

In the normal case, one or more characters in the system-generated character array will not be needed, and the elements of A$() occupied by the unwanted characters can be used for new characters. In this instance, it generally is not necessary to alter the dimensioned sizes of A$() or C$(). To store the new code sequence for each new character, one of the assignment statements in PLOT001A must be changed to specify the new codes. The procedure is as follows:

1. Load and run the Start Module.

2. Depress Special Function Key 0, instructing the system to initialize a Character Generation Array. Select either the Standard Character array or the Enhanced Character array. The Start Module will automatically call in either the PLOT001A or PLOT002A Routine to create the array.

3. When the prompt "ENTER THE NUMBER OF THE DESIRED OUTPUT ADDRESS" appears, key RESET.

4. Enter LIST S to list the first 15 lines of PLOT001A. Go through the assignment statements for A$() in PLOT001A, substituting new code sequences for exisitng code sequences where desired. Remember that characters defined by the deleted code sequences can no longer be plotted from this array. Remember, too, that a maximum of 23 plot instructions (each represented by a two-digit hex code) can be stored in each element of A$().

5. When all new code sequences have been inserted, key RESET, RUN, (EXEC). The prompt "ENTER THE NUMBER OF THE DESIRED OUTPUT ADDRESS" is displayed once again. Respond to this prompt, and to the subsequent prompt requesting a file name, in the normal fashion. The modified character array will be saved on disk, and may be recalled (with the Load Character Generation Array Routine) whenever needed.

NOTE:

Character I.D. codes must be stored in ascending sequence in A$(). The codes do not need to be consecutive, but they must be in ascending order.

C.6 INCREASING THE SIZE OF THE CHARACTER GENERATION ARRAY

Although it is not generally necessary to alter the size of the character array, there are special circumstances in which alteration of the array size is necessary:

1. If the new characters are so elaborate or complex as to require more than 23 plotter movements. In this case, the element length of A$() must be increased to allow for the additional plot codes. (The maximum length is 64 bytes; each byte stores one plot instruction, represented as a two-digit hex code.)

2. If more than 63 (or 89) characters are to be stored in the character array. In this case, the number of elements in A$() must be increased, since each element contains the code sequence for one character. (The maximum number of elements in a one-dimensional array is 255. As a practical matter, however, the size of the array is limited by the maximum number of unique character I.D. codes which are available, since each character stored in the array must be identified with a unique I.D. code. The legal range of hex codes which can be used to create I.D. codes is from HEX(21) to HEX(9F) inclusive. This range contains a total of 127 unique codes. Therefore, the maximum number of characters which can be defined is 127. When the space, which is not defined in the character array, is included, the maximum size of the character set in one array is 128.

Data is packed from A$() element by element into C$(). This procedure utilizes a loop with a counter which specifies the number of elements to be processed from A$(). Numeric variable 'A' serves as the counter. Its value is assigned at line 40 of PLOT001A (or PLOT002A). In the standard version, A=63, since A$() contains 63 elements. If the number of elements in A$() is increased, the value of the counter must be increased correspondingly; otherwise, only the first 63 elements of A$() will be packed into C$(). For example, if A$() is dimensioned to 72 elements, line 40 must be changed to A=72.

Both the creator array, A$(), and the packed array, C$(), are dimensioned at line 20 of the PLOT001A and PLOT002A routines. C$() is a packing array which when filled, saves the array on disk, retaining excess bytes and is ready for further packing. Hence there is no need to alter C$() however much A$() is altered.

In disk versions of the Utilities, the PLOT001A and PLOT002A routines record the packed array C$() in a cataloged data file on disk (the file name and location re specified by the operator). Prior to packing the required size is determined by the routine and the opened file is automatically proportioned accordingly.

The character generation array must be recalled from disk into memory by the Load Character Generation Array Routine prior to plotting any characters. Array C$() is used to store the characters when they are loaded from disk and must be large enough to accommodate all the array. Line 20 of the dimension routine contains a DIM statement which dimensions C$(), among other variables. The DIM statement can be changed to increase the size of C$(). If C$() is not redimensioned in the utility set, only part of the character data file on disk will be loaded into memory for plotting. During normal operation, the user is required to select the dimension routines which contain the appropriately sized C$() and may change the size accordingly.

C.7 RESTRICTIONS

The programmer should read and familiarize himself with the following list of restrictions before attempting to create new characters:

1. The total number of vertices used to define a character may not exceed 23 in the standard or enhanced character array. Elements in A$() are dimensioned to 24 bytes in length, with the first byte reserved for the character I.D. code, and the remaining 23 bytes available for plotting codes. If more than 23 vertices are required to define a character, the element length of A$() can be increased up to a maximum of 64 bytes. Such an increase entails increasing the size of the packed array C$() correspondingly.

2. The lowest legal value for a Character I.D. code is HEX(81). I.D. codes lower than HEX(81) are not allowed. Therefore, the lowest hex code which can be used to create a character I.D. code is HEX(21), since HEX(21) ADDED to HEX(60) yields HEX(81).

3. The highest legal value for a character I.D. code is HEX(FF). I.D. codes higher than HEX(FF) are not allowed. Therefore, the highest hex code which can be used to create a character I.D. code is HEX(9F), since HEX(9F) ADDed to HEX(60) yields HEX(FF).

4. The total number of characters defined in the standard character array is 63 ;89 for the enhanced array. If more characters must be stored in the array, the number of elements in A$() can be increased from 63 (or 89) to a maximum of 255. (Note, however, that since the valid range of character I.D. codes is from 81 to FF, a total of 127 unique codes is available. Thus a total of 127 characters can be defined in one array, limiting the practical maximum array size to 127 elements.) An increase in the size of A$() is automatically taken care of by C$() and require no adjustments. The numeric variable 'A' contains the counter, which is set at 63 in line 40 of PLOT001A (set of 89 in line 40 of PLOT002A). If A$() is given additional elements, the counter must be changed to reflect the number of elements in A$().

NOTE:

A$() may not be changed to a two-dimensional array.

5. The character I.D. codes must be stored in ascending sequence in the character array. Character I.D. codes need not be consecutive, but they must be sequential.

6. The last coded plot instruction in a plot instruction sequence may not be a 20 ("plot to 20"). The 20 code is legal anywhere within a sequence of instructions, but its presence as the last code in the sequence has a special significance to the subroutine, and is not permitted.

7. The space reserved for C$() must contain at least one more byte than will be occupied by the elements of C$(). The characters are set apart by I.D. codes in C$ and when selecting a string of instructions for a character, the routine looks for the beginning and ending I.D. codes. To accomplish this for the final letter, C$() is initialized with FF. Thus when searching for the final I.D. codes, the routine will encounter FF and establish the final string of instruction. If C$() is completely filled by instructions however, ther is no final I.D. code and an error will be indicated.

8. In disk versions of the Plotter Utilities, the size of the cataloged data file in which the character array is stored is automatically increased if the character array size is increased.

APPENDIX D: RESERVED VARIABLE LIST

PAGE 1 OF 1

Wang BASIC Variable Check-off List

PROGRAM NAME PLOTTER SUBROUTINES DATE 3/10/78

VERSION PROGRAMMER

SYSTEM PLOTTER UTILITIES

NUMERIC SCALARS
FORMAT = MN

N\M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ	0	0	0	0	0	0																				
1	0	0	0	0	0	0																				
2	0	0	0	0	0	0																				
3			0	0	0	0																				
4			0	0	0	0																				
5			0	0	0																					
6			0	0	0																					
7	●	0	0	0	0																					
8	0	0	0		0																					
9	0	0	0		0																					
0	0	0	0																							

NUMERIC ARRAYS
FORMAT = MN(

N\M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ	0	0			0																					
1																										
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
0																										

ALPHA NUMERIC SCALARS
FORMAT = MN$

N\M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ	0	0	0		0																					
1		0		0																						
2			0																							
3																										
4																										
5																										
6																										
7																										
8																										
9																										
0			0																							

ALPHA NUMERIC ARRAYS
FORMAT = MN$(

N\M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Δ			0	0																						
1			0																							
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
0			0																							

NOTE:
0 = NON COMMON
1 = COMMON DEFINED BY THIS MODULE
2 = COMMON DEFINED BY PREVIOUS MODULE

APPENDIX E:
LOADING PROGRAM OVERLAYS WITH THE PLOTTER UTILITIES

If the plotter utilities are to be used in an application program which calls in program overlays from disk or tape, it will be necessary to COM all variables used by the utilities. Non-common variables are automatically cleared by the LOAD statement when a program overlay is called into memory.

All dimensioned variables can be changed from non-common to common by replacing the DIM at line 20 of the dimension routines with a COM. In addition, the following scalar numeric variables must be defined as common variables. These variables are not defined in the DIM statement on line 20, but they are used by the subroutines to store critical internal pointers, and must be defined as common when overlays are used:

```
A0
B0
A8
B8
A9
B9
D8
D9
```

APPENDIX F: USING THE HARDWARE CHARACTER SET IN CONJUNCTION WITH THE PLOTTER UTILITIES

The Models 2202, 2212, 2272, 2281P, and 2282 provide their own built-in hardware character sets which are independent of the Plotter Utilities character set. The hardware character set in each case is generated by the plotter microprocessor, and does not require any supporting software in the System 2200. (Only the Model 2232B Flatbed Plotter does not provide a built-in hardware character set; on that plotter, all character generation must be accomplished with supporting software.) In general, it is easier and more convenient to use the Plotter Utilities character set when plotting with the utility subroutines. In rare cases, however, the hardware character set may be the preferred choice. (One such case is that of a configuration built around a System 2200S without Option 23 or 24. Since a 2200S with neither of those options does not have the Sort statements, it cannot support the character generation routine in the Plotter Utilities, which requires a Sort capability. In this case, therefore, the programmer must rely on the hardware character set.)

The principal problem involved in labeling with the hardware character set when plotting with the plotter utilities is that the characters must be plotted "outside" the utilities. That is, the hardware characters are created with a PLOT statement independent of the plotter utility routines, and the plotter movement which results is not monitored by the subroutines. Thus, the physical position of the plotter at the conclusion of the labeling operation no longer corresponds to the position indicated by the internal pointers of the plotter subroutines. If the application program is written so that all plotting is completed before any labeling begins, this disparity does not present a problem. If, however, the application program requires some additional plotting after labeling, a serious problem arises. In this case, the programmer must see to it that the physical plotter position coincides with the plotter position assumed by the utility subroutines before proceeding with any additional plotting.

To guarantee that the physical plotter position will correspond to the utilities' assumed plotter position, the plotter must be returned to its original position (the last position recognized by the utility subroutines) following completion of the labeling operation. For all plotters except the Model 2272, the following PLOT statement can be used to return the plotter to the start of a plotted character string:

```
PLOT <-X*LEN(A$),-Y*LEN(A$),>
```

X and Y are the horizontal and vertical spaces, respectively, and A$ contains the plotted character string.

For the Model 2272, the algorithm must be modified as follows:

```
PLOT <2*(-X)*LEN(A$),2*(-Y)*LEN(A$),>
```

In this case, again, X and Y contain the horizontal and vertical spacing, and A$ holds the plotted character string.

If the application involves labeling several different points before plotting is resumed, the recommended procedure is to back up to the starting point of each string after it is plotted, and moved to the next labeling point with the plotter utilities.

APPENDIX G: GIN MODE ROUTINE FOR TEKTRONIX GRAPHIC TERMINAL (DEFFN' 23)

G.1 PROGRAM DESCRIPTION

Certain graphic display terminals in the Tektronix 4000 series offer a special feature called Graphic Input (GIN) Mode. In GIN Mode, the X and Y coordinates of the terminal display crosshairs can be transmitted to a receiving system--in this case, the System 2200. (Note that only selected graphic terminal models offer the GIN Mode feature; consult the Tektronix reference literature provided with your terminal to determine whether this feature is supported.)

The GIN Mode feature permits an operator to obtain the coordinates of any point on the graphic display by manually positioning the crosshairs at the desired point; in effect, GIN Mode enables the display to function as a digitizer. A special GIN Mode Routine (DEFFN' 23) provided in the Plotter Utilities Package enables the 2200 to accept, interpret, and display the coordinates transmitted by the graphic terminal.

G.2 SUBROUTINE ARGUMENT LIST

```
GOSUB' 23
```

The GIN Mode Routine has no arguments. Execution of the GOSUB' 23 statement automatically places the display terminal in GIN Mode, and readies the 2200 to accept coordinates transmitted by the terminal.

Once GOSUB' 23 has been executed, graphic input is controlled from the terminal keyboard according to the following procedure:

1. Manually position the crosshairs to a desired point on the screen.

2. Touch any key on the terminal keyboard. The character entered is referred to as the "flag character;" it may be used to signal special conditions to the controlling BASIC program. (Do <u>not</u> use the BREAK or ESC keys for this purpose.)

3. Touch the RETURN key on the terminal keyboard. When RETURN is keyed, the X and Y coordinates of the current crosshair position are sent to the 2200, along with the flag character, and the terminal is taken out of GIN Mode. The information received by the 2200 is stored in the following variables:

A7	=	X coordinate of crosshairs.
B7	=	Y coordinate of crosshairs.
CO$	=	Flag character.

Coordinates are expressed in Tektronix Graphic Display Units (GDU's). Note that the flag character has no functional purpose; it is simply a convenient means of signalling special conditions (such as first point, last point, etc.) to the controlling 2200 program.

Because the GIN Mode is terminated when RETURN is keyed, the GOSUB' 23 statement must be re-executed each time a new set of screen coordinates is to be transmitted. Like the other Plotter Utility Routines, GOSUB' 23 can be executed under program control, or directly from the keyboard with Special Function Key 23.

APPENDIX H: KEYBOARD/CHARACTER SET CROSS - REFERENCE

KEYBOARD	CHARACTER SET	KEYBOARD	CHARACTER SET	KEYBOARD	CHARACTER SET
A	A	0	Ø	i	⏚
B	B	!	!	j	⏊
C	C	@	@	k	Λ
D	D	#	#	l	λ
E	E	$	$	m	μ
F	F	%	%	n	Y
G	G	↑	↑	o	Ω
H	H	&	&	p	π
I	I	*	*	q	Ψ
J	J	(	(	r	ρ
K	K	)	)	s	σ
L	L	-	-	t	θ
M	M	+	+	u	Σ
N	N	=	=	v	⟂
O	O	<	<	w	≣
P	P	>	>	x	⌓
Q	Q	:	:	y	⌓
R	R	;	;	z	⌓
S	S	"	"		
T	T	/	/		
U	U	?	?		
V	V	.	.		
W	W	,	,		
X	X	_*	←		
Y	Y	_*	[		
Z	Z	_*	\		
1	1	_*	]		
2	2	a	α		
3	3	b	β		
4	4	c	Δ		
5	5	d	δ		
6	6	e	∈		
7	7	f	Φ		
8	8	g	Γ		
9	9	h	╪		
'	'				

*Use HEX Code

INDEX

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-3838D

TITLE OF MANUAL **2200 PLOTTER UTILITIES MANUAL**

COMMENTS:

Fold

Fold

Name ______________________ Title ______________________
Company ______________________ Tel. # ______________________
City ______________ State ______________ Zip Code ______________

(Please tape, Postal regulations prohibit the use of staples.)

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Corporate Publications Department

Fold

Cut along dotted line.

Printed in U.S.A.
13-1019A

International Representatives

Argentina
Bahamas
Bahrain
Bolivia
Botswana
Brazil
Canary Islands
Chile
Colombia
Costa Rica
Cyprus
Denmark
Dominican Republic
Ecuador
Egypt
El Salvador
Finland
Ghana
Greece
Guam
Guatemala
Haiti
Honduras
Iceland
India
Indonesia
Ireland
Israel
Italy
Ivory Coast
Japan
Jordan
Kenya
Korea
Kuwait
Lebanon
Liberia
Malaysia
Malta
Mexico
Morocco
New Guinea
Nicaragua
Nigeria
Norway
Paraguay
Peru
Philippines
Portugal
Qatar
Saudi Arabia
Scotland
Senegal
South Africa
Spain
Sri Lanka
Sudan
Tasmania
Thailand
Turkey
United Arab Emirates
Uruguay
Venezuela
Zimbabwe

United States

Alabama
Birmingham
Mobile
Alaska
Anchorage
Juneau
Arizona
Phoenix
Tucson
California
Anaheim
Burlingame
Culver City
Emeryville
Fountain Valley
Fresno
Los Angeles
Sacramento
San Diego
San Francisco
Santa Clara
Ventura
Colorado
Englewood
Connecticut
New Haven
Stamford
Wethersfield
District of Columbia
Washington

Florida
Coral Gables
Hialeah
Hollywood
Jacksonville
Miami
Orlando
Sarasota
Tampa
Georgia
Atlanta
Savannah
Hawaii
Honolulu
Maui
Idaho
Boise
Illinois
Arlington Heights
Chicago
Morton
Oakbrook
Park Ridge
Rock Island
Rosemont
Springfield
Indiana
Fort Wayne
Indianapolis
South Bend

Iowa
Ankeny
Kansas
Overland Park
Wichita
Kentucky
Louisville
Louisiana
Baton Rouge
Metairie
Maine
Portland
Maryland
Baltimore
Bethesda
Gaithersburg
Rockville
Massachusetts
Boston
Burlington
Chelmsford
Lawrence
Littleton
Lowell
Methuen
Tewksbury
Worcester
Michigan
Grand Rapids
Kalamazoo
Lansing

Southfield
Minnesota
Eden Prairie
Minneapolis
Mississippi
Jackson
Missouri
Creve Coeur
St. Louis
Nebraska
Omaha
Nevada
Las Vegas
New Hampshire
Manchester
New Jersey
Bloomfield
Clifton
Edison
Mountainside
Toms River
New Mexico
Albuquerque
Santa Fe
New York
Albany
Jericho
Lake Success
New York City
Rochester

Syossett
Syracuse
Tonawanda
North Carolina
Charlotte
Greensboro
Raleigh
Ohio
Akron
Cincinnati
Cleveland
Independence
Toledo
Worthington
Oklahoma
Oklahoma City
Tulsa
Oregon
Eugene
Portland
Salem
Pennsylvania
Allentown
Erie
Harrisburg
Philadelphia
Pittsburgh
State College
Wayne
Rhode Island
Providence

South Carolina
Charleston
Columbia
Tennessee
Chattanooga
Knoxville
Memphis
Nashville
Texas
Austin
Dallas
El Paso
Houston
San Antonio
Utah
Salt Lake City
Virginia
Newport News
Norfolk
Richmond
Rosslyn
Springfield
Washington
Richland
Seattle
Spokane
Wisconsin
Appleton
Brookfield
Green Bay
Madison
Wauwatosa

International Offices

Australia
Wang Computer Pty., Ltd.
Adelaide, S.A.
Brisbane, Qld.
Canberra, A.C.T.
Perth, W.A.
South Melbourne, Vic 3
Sydney, NSW
Austria
Wang Gesellschaft, m.b.H.
Vienna
Belgium
Wang Europe, S.A.
Brussels
Erpe-Mere
Canada
Wang Canada Ltd.
Burlington, Ontario
Burnaby, B.C.
Calgary, Alberta
Don Mills, Ontario
Edmonton, Alberta
Halifax, Nova Scotia
Hamilton, Ontario
Montreal, Quebec
Ottawa, Ontario
Quebec City, Quebec
Toronto, Ontario

Victoria, B.C.
Winnipeg, Manitoba
China
Wang Industrial Co., Ltd.
Taipei
Wang Laboratories, Ltd.
Taipei
France
Wang France S.A.R.L.
Paris
Bordeaux
Lille
Lyon
Marseilles
Nantes
Nice
Rouen
Strasbourg
Great Britain
Wang (U.K.) Ltd.
Richmond
Birmingham
London
Manchester
Hong Kong
Wang Pacific Ltd.
Hong Kong

Japan
Wang Computer Ltd.
Tokyo
Netherlands
Wang Nederland B.V.
IJsselstein
Groningen
New Zealand
Wang Computer Ltd.
Auckland
Christchurch
Wellington
Panama
Wang de Panama (CPEC) S.A.
Panama City
Puerto Rico
Wang Computadoras, Inc.
Hato Rey
Singapore
Wang Computer (Pte) Ltd.
Singapore
Sweden
Wang Skandinaviska AB
Stockholm
Gothenburg

Malmö
Switzerland
Wang A.G.
Zurich
Basel
Bern
Geneva
Lausanne
St. Gallen
Wang Trading A.G.
Zug
West Germany
Wang Deutschland, GmbH
Frankfurt
Berlin
Cologne
Düsseldorf
Essen
Freiburg
Hamburg
Hannover
Kassel
Mannheim
Munich
Nürnberg
Saarbrücken
Stuttgart

WANG

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

Printed in U.S.A.
700-3838D
3-82-5C