CHAPTER 16

The GFONT Procedure

Overview

The GFONT procedure displays new or existing fonts and creates user-generated fonts for use in SAS/GRAPH programs. These fonts can contain standard Roman alphabet characters, foreign language characters, symbols, logos, or figures.

The GFONT procedure

- ☐ displays SAS/GRAPH software fonts
- ☐ displays fonts that were previously generated with the GFONT procedure (user-generated fonts)
- ☐ displays the character codes or hexadecimal values that are associated with the characters in a font
- ☐ creates stroked fonts or polygon fonts.

Each of these activities has its own requirements, its own process, and its own options (although some options are valid for either process). In this chapter, each topic to which this distinction applies is divided into two sections: "Displaying Fonts" and "Creating Fonts."

About Displaying Fonts

You can use the GFONT procedure to display a font when you want to do one of the following:

- □ review the characters that are available in either Institute-supplied fonts or user-generated fonts
- □ see the character codes or the hexadecimal values that are associated with the characters in a font.

When you display a font, you can modify the color and height of displayed font characters, draw reference lines around the characters, or display the associated character codes or hexadecimal values. See Example 1 on page 698.

About Creating Fonts

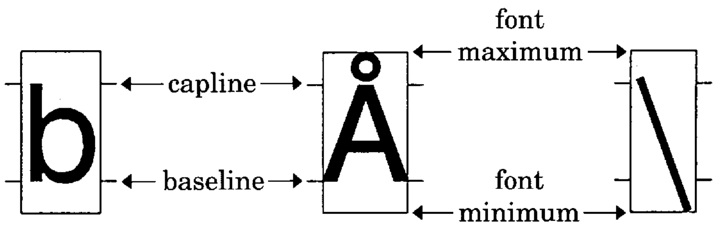
You can use the GFONT procedure to create and store fonts of your own design. The GFONT procedure is not limited to creating alphabet fonts. You can use it to create and store any series of figures that you can draw using X and Y coordinates or that you can digitize. The characters or figures in a font can be displayed with any SAS/GRAPH statement or option that allows for font specification and a text string (for example, a TITLE statement). See “Creating a Font” on page 687 for details.

Concepts

About Fonts

Some specialized terms are associated with font characteristics. The *capline* of a font is the highest point of a normal uppercase letter. The *baseline* is the line upon which the characters rest. The *font maximum* is the highest vertical coordinate in a font. The *font minimum* is the lowest vertical coordinate in a font. Figure 16.1 on page 676 illustrates these GFONT procedure terms:

Figure 16.1 Parts of a Font



Specialized terms also exist for types of fonts. The term *uniform font* refers to a font in which all of the characters occupy exactly the same amount of space, even though the characters themselves are different sizes. Each character in a uniform font is placed in the center of its space, and a fixed amount of space is added between characters. A *proportional font* is a font in which each character occupies a space that is proportional to its actual width (for example, m occupies more space than i). The characters in a *stroked font* are drawn with discrete line segments or circular arcs. Figure 16.2 on page 677 illustrates a stroked font with several characters from the Simplex font.

Figure 16.2 Characters from a Stroked Font

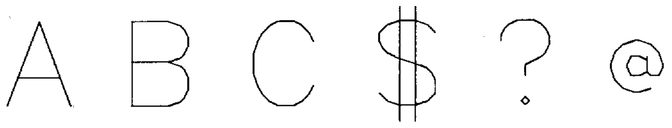
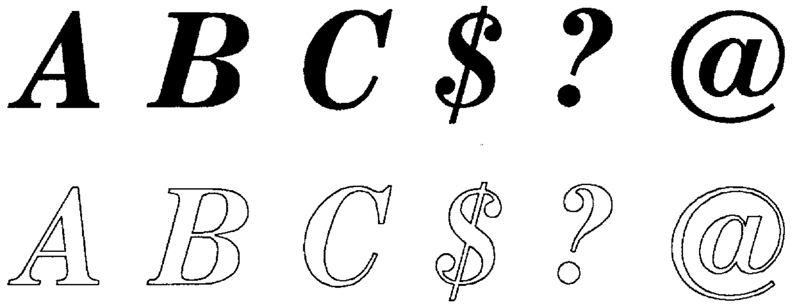


Figure 16.3 on page 677 illustrates two types of *polygon fonts*: filled (CENTBI) and outline (CENTBIE). A *filled font* is a polygon font in which the areas between the lines are solid. An *outline font* is a polygon font in which the areas are empty.

Figure 16.3 Filled and Outline Characters from Polygon Fonts



All font characters, regardless of whether they are stroked or polygon, are drawn with line segments. In the GFONT procedure, the term *line segment* means a continuous line that can change direction. For example, the letter C in Figure 16.2 on page 677 is drawn with one line segment, while the letter A can be drawn with two.

Polygon characters can also be drawn with one or more line segments. In a polygon font, one character can be made up of a single polygon, multiple polygons, or polygons with holes. For example, the letter C in Figure 16.3 on page 677 is a single polygon with one line segment. The question mark (?) is made up of two polygons, each drawn with a separate line segment. The letter A is one polygon with a hole in it. It is drawn with one line segment that is broken to form the outer boundary of the figure and the boundary of the hole.

About the Libref GFONTO

The GFONT procedure stores user-generated fonts in the location that is associated with the libref GFONT0. Therefore, before you create a font or display a user-generated font, you must submit a LIBNAME statement that associates the libref GFONT0 with the location where the font is to be stored, as follows:

```
libname gfont0 'SAS-data-library';
```

Since the GFONT0 library is the first place that SAS/GRAPH software looks for fonts, you should always assign that libref to the library that contains your personal fonts. If for some reason you have personal fonts in more than one SAS data library, assign them librefs in the sequence GFONT0, GFONT1, GFONT2, and so forth. The search for entries terminates if there is a break in the sequence; the catalog GFONT1.FONTS is not checked if the libref GFONT0 is undefined. If the libref GFONT0 is not defined, by default SAS/GRAPH software begins searching for fonts in SASHELP.FONTS.

To cancel or redefine the libref GFONT*n*, submit the following statement:

```
goptions reset=all fcache=0;
```

Note that when you specify RESET=ALL, all graphics options are reset to their default values. Once you have cleared the font cache, you can redefine the libref with another LIBNAME statement.

Procedure Syntax

Requirements: A font name is required. To display a font, include NOBUILD. To create a font, include DATA=.

Global statements: FOOTNOTE, TITLE

Reminder: The procedure can include the SAS/GRAPH NOTE statement.

Supports: Output Delivery System (ODS)

PROC GFONT NAME=*font-name*
  mode
  <*display-option(s)*>
  <*creation-option(s)*>;

PROC GFONT Statement

The PROC GFONT statement can either create user-defined fonts or display existing software fonts. Therefore, it names the font to be created or displayed. If the procedure creates a font it names the input data set. Optionally, the procedure modifies the design and appearance of the fonts that you create or display, and specifies a destination catalog for graphics output.

Syntax

PROC GFONT NAME=*font-name*
  mode
  <*display-option(s)*>
  <*creation-option(s)*>;

- *mode* must be one of the following:

  DATA=*font-data-set*

  NOBUILD

- *display-option(s)* can be one or more of the following:

  CTEXT=*text-color*

  GOUT=<*libref.*>*output-catalog*

  HEIGHT=*character-height*<*units*>

  NOKEYMAP

  NOROMAN

  NOROMHEX

  REFCOL=*reference-line-color*

  REFLINES

  ROMCOL=*code-color*

  ROMFONT=*font*

  ROMHEX

  ROMHT=*height*<*units*>

  SHOWALL

  SHOWROMAN

- *creation-option(s)* can be one or more of the following:

  BASELINE=*y*

  CAPLINE=*y*

  CHARSPACETYPE=DATA | FIXED | NONE | UNIFORM

  CODELEN=1 | 2

  FILLED

  KERNDATA=*kern-data-set*

  MWIDTH=*character-width*

  NODISPLAY

  NOKEYMAP

  RESOL=1...4

  ROMHEX

  SHOWROMAN

  SPACEDATA=*space-data-set*

  UNIFORM

For more detail on using the GFONT syntax, see “Displaying Fonts: Required Arguments, Options” on page 679 and “Creating Fonts: Required Arguments, Options ”on page 682.

Displaying Fonts: Required Arguments, Options

Required Arguments for Displaying Fonts

NAME=*font-name*
N=*font-name*

specifies the font to be displayed. *Font-name* can be the name of a SAS software font or a font you previously created.

See also: Chapter 6, “SAS/GRAPH Fonts,” on page 125

NOBUILD
NB
specifies that the GFONT procedure is to display an existing font. The NOBUILD argument tells the procedure that no font is being generated and not to look for an input data set.

Featured in: Example 1 on page 698

To display a user-generated font, you must define libref GFONT0. See "About the Libref GFONT0" on page 677 for details.

Options for Displaying Fonts

Options that can be used for either font display or font creation are described here and in "Options for Creating Fonts" on page 683.

Options that display a font can be used when you create a font if you also display it (that is, the NODISPLAY option is not used in the PROC GFONT statement). However, none of the display options affect the design and appearance of the stored font except the NOKEYMAP, SHOWROMAN, and ROMHEX options.

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points

If you omit *units*, a unit specification is searched for in this order:

1 the value of GUNIT= in a GOPTIONS statement
2 the default unit, CELLS.

CTEXT=*text-color*
CT=*text-color*
specifies a color for the body of the characters. If you do not use the CTEXT= option, a color specification is searched for in the following order:

1 the CTEXT= option in a GOPTIONS statement
2 the default, the first color in the colors list.

The CTEXT= value is not stored as part of the font.

Featured in: Example 2 on page 700

GOUT=<*libref.*>*output-catalog*
specifies the SAS catalog in which to save the graphics output produced by the display of the font. The GOUT option is ignored if you use the NODISPLAY option in the PROC GFONT statement. You can use the GREPLAY procedure to view the output that is stored in the catalog. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: "Storing Graphics Output in SAS Catalogs" on page 49

HEIGHT=*character-height*<*units*>
H=*character-height*<*units*>
specifies the height of the font characters in number of units, *n*. Height is measured from the minimum font measurement to the capline. By default, HEIGHT=2.

Featured in: Example 1 on page 698

NOKEYMAP

specifies that the current key map is ignored when displaying the font and its character codes or hexadecimal values. If you do not use the NOKEYMAP option when you display a font, the current key map remains in effect. If any characters in the font are not available through the current key map, they are not displayed and a warning is issued in the SAS log. This happens when the key map is asymmetrical, that is, not all characters in the font are mapped into the current key map.

Displaying a font using the NOKEYMAP option enables you to see all of the characters in the font, including those that are not mapped into your current key map. Note that only those characters that are mapped into your current key map are available (that is, those that are displayed when you display the font without the NOKEYMAP option).

See also: Chapter 6, "SAS/GRAPH Fonts," on page 125 Chapter 18, "The GKEYMAP Procedure," on page 719 and the NOKEYMAP option on page 685 for Creating Fonts

NOROMAN
NR

turns off the automatic display of character codes that are produced when you use the SHOWROMAN option during font creation.

NOROMHEX
NOHEX

turns off the automatic display of hexadecimal values that are produced when you use the ROMHEX option during font creation.

REFCOL=*reference-line-color*

specifies a color for reference lines. By default, the first color in the colors list is used.

REFLINES

draws reference lines around each displayed character. Vertical reference lines show the width of the character. Horizontal reference lines show the font maximum and the font minimum, as well as the baseline and the capline. See Figure 16.1 on page 676 for an illustration of the placement of reference lines.

ROMCOL=*code-color*
RC=*code-color*

specifies the color of the character codes or hexadecimal values that are displayed with the SHOWROMAN and ROMHEX options. If you do not use the ROMCOL= option, a color specification is searched for in the following order:

1 the CTEXT= option in a GOPTIONS statement

2 the default, the first color in the colors list.

The ROMCOL= value is not stored as part of the font.

Featured in: Example 1 on page 698

ROMFONT=*font*
RF=*font*

specifies the font for character codes and hexadecimal values that are displayed by the SHOWROMAN and ROMHEX options. If you do not use the ROMFONT= option, a font specification is searched for in the following order:

1 the FTEXT= option in a GOPTIONS statement

2 the default hardware font, NONE.

Featured in: Example 1 on page 698

ROMHEX
HEX
displays hexadecimal values below the font characters. If you use both the ROMHEX and SHOWROMAN options, both the character codes and the hexadecimal values are displayed. You also can use the ROMHEX option when you create a font.

See also: the ROMHEX option on page 686

ROMHT=*height*<*units*>
RH=*height*<*units* >
specifies the height of the character codes and the hexadecimal values that are displayed with the SHOWROMAN and ROMHEX options in number of units, *n*. If you do not use the ROMHT= option, a height specification is searched for in the following order:

1. the HTEXT= option in a GOPTIONS statement
2. the default, ROMHT=1.

Featured in: Example 1 on page 698

SHOWALL
displays the font with a space for every possible character position whether or not a font character exists for that position. The characters that are displayed are those available under your current key map, unless you use the NOKEYMAP option. The SHOWALL option usually is used in conjunction with the ROMHEX option, in which case all possible hexadecimal values are displayed. If, under your current key map, a font character is available for a position, it displays above the hexadecimal value. If no character is available for a position, the space above the hexadecimal value is blank. You can use the SHOWALL option to show where undefined character positions fall in the font.

SHOWROMAN
SR
displays character codes below the font characters even if they are not displayed automatically with the font. If you use both the SHOWROMAN and ROMHEX options, both the character codes and the hexadecimal values are displayed. You can also use the SHOWROMAN option when you create a font.

See also: the SHOWROMAN option on page 686 for Creating Fonts.

Featured in: Example 1 on page 698

Details

To display a font, you must specify the name of the font with the NAME= argument and include the NOBUILD argument. For example, to display the Weather font with character codes that are displayed in the Swiss font, use the following statement:

```
proc gfont name=weather nobuild romfont=swiss;
```

Creating Fonts: Required Arguments, Options

Required Arguments for Creating Fonts

NAME=*font-name*
N=*font-name*
assigns a name to the font that you create. *Font-name* is the name of a catalog entry and must be a valid SAS name of no more than eight characters. Do not use the name of an Institute-supplied font or NONE for the name of a font.

Featured in: Example 2 on page 700

DATA=*font-data-set*

specifies the SAS data set that the GFONT procedure uses to build the font. The data set must be sorted by the variables CHAR and SEGMENT. By default, the procedure uses the most recently created data set as the font data set.

See also: "SAS Data Sets" on page 25

Featured in: Example 2 on page 700

When you create a font, you must define the libref GFONT0. See "About the Libref GFONT0" on page 677 for details.

Note: If a user-generated font has the same name as an Institute-supplied font and if the libref GFONT0 has been defined, the user-generated font is used because GFONT0 is searched first. △

Options for Creating Fonts

Options that can be used for either font display or font creation are described here and in "Options for Displaying Fonts" on page 680.

Options that display a font can be used when you create a font if you also display it (that is, the NODISPLAY option is not used in the PROC GFONT statement). However, none of the display options affect the design and appearance of the stored font except the NOKEYMAP, SHOWROMAN, and ROMHEX options.

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points

If you omit *units*, a unit specification is searched for in this order:

1 the value of GUNIT= in a GOPTIONS statement

2 the default unit, CELLS.

BASELINE=*y*
B=*y*

specifies the vertical coordinate in the font data set that is the baseline of the characters. The baseline is the line upon which the letters rest. If you do not use the BASELINE= option, the GFONT procedure uses the lowest vertical coordinate of the first character in the font data set.

CAPLINE=*y*
C=*y*

specifies the vertical coordinate in the font data set that is the capline of the characters. The capline is the highest point of normal Roman capitals. If you do not use the CAPLINE= option, the GFONT procedure uses the highest vertical coordinate in the font data set, in which case the capline and the font maximum are the same. See Figure 16.1 on page 676 for an illustration of capline and font maximum.

If you use the CAPLINE= option, then when the GFONT procedure calculates the height of a character, any parts of the character that project above the capline are ignored in the calculation.

You can use this option to prevent an accented capital like A from being shortened to accommodate the accent. For example, if you do not use the CAPLINE= option, the capline and the font maximum are the same and the A is shortened to make room for the accent below the capline. However, if CAPLINE= is used, the top of the letter A is at the capline, and the accent is drawn above the capline and below the font maximum.

CHARSPACETYPE=DATA | FIXED | NONE | UNIFORM
CSP=DATA | FIXED | NONE | UNIFORM

specifies the type of intercharacter spacing. The following are valid values:

DATA

specifies that the first observation for each character sets the width of that character. When CHARSPACETYPE=DATA, the PTYPE variable is required, and the observation that specifies the width of the character must have a PTYPE value of W. See "The Font Data Set" on page 687 for details on the PTYPE variable.

Intercharacter spacing is included in the character's width. For example, if the first observation for the letter A specifies a character width of 10 units and the A itself occupies only 8 units, the remaining 2 units serve as intercharacter spacing.

Note: The character can extend beyond the width that you specified in the first observation if desired. △

FIXED

adds a fixed amount of space between characters based on the font size. The width of the individual character is determined by the data that generate the character.

NONE

specifies that no space is added between characters. The width of the individual character is determined by the data that generate the character. This type of spacing is useful for script fonts in which the characters should appear connected.

UNIFORM

specifies that the amount of space that is used for each character is uniform rather than proportional. This means that each character occupies the same amount of space. For example, in uniform spacing the letters m and i occupy the same amount of space, whereas in proportional spacing m occupies more space than i. In uniform spacing, the character is always centered in the space and a fixed space is added between characters.

When UNIFORM is specified, the amount of space that is used for each character is one of the following:

- by default, the width of the widest character in the font.
- the width specified by the MWIDTH= option. See the MWIDTH= option on page 685 for details.

Specifying CHARSPACETYPE=UNIFORM is the same as using the UNIFORM option.

Note: By default, CHARSPACETYPE=FIXED. △

CODELEN=1 | 2

specifies the length in bytes of the CHAR variable. By default, CODELEN=1. To specify double-byte character sets for languages such as Chinese, Japanese, or Korean, use CODELEN=2. If you specify a double-byte character set, you cannot specify kerning or space adjustment with the KERNDATA= or SPACEDATA= options.

FILLED
F

specifies that the characters in a user-generated polygon font are filled.

Featured in: Example 2 on page 700

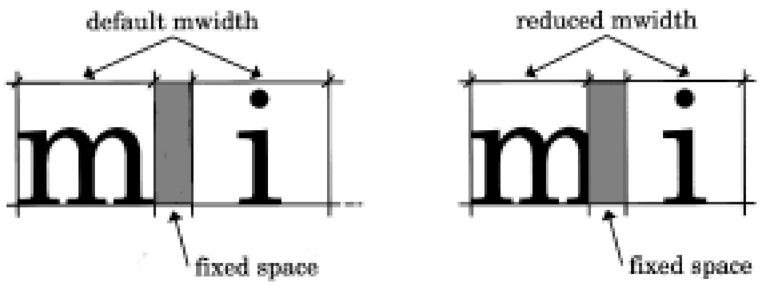
KERNDATA=*kern-data-set*
KERN=*kern-data-set*
specifies the SAS data set that contains kerning information. When the KERNDATA= option is used during font creation, the data that are contained in the kern data set are applied to the font and stored with it. You cannot specify kerning for a double-byte character set that is created by using the option CODELEN=2.

See also: "The Kern Data Set" on page 694

MWIDTH=*character-width*
specifies the width of a character in a uniform font, where *character-width* is the number of font units. The MWIDTH= option is only valid when you specify uniform spacing by using the UNIFORM option or when you specify CHARSPACETYPE=UNIFORM. If you do not use MWIDTH=, the default is the width of the widest character in the font (usually the letter m).

Typically, you use the MWIDTH= option to tighten the spacing between characters. To do this, specify a smaller value (narrower width) for *character-width*. Figure 16.4 on page 685 shows the effect of decreasing the space that is allowed for uniformly spaced characters.

Figure 16.4 Using the MWIDTH= Option to Modify Spacing



See also: the CHARSPACETYPE= option on page 684 and the UNIFORM option on page 687

NODISPLAY
ND
specifies that the GFONT procedure is not to display the font that it is creating.

NOKEYMAP
specifies that the current key map is ignored when you generate and use the font that is being created, and that the character codes you enter are not mapped in any way before being displayed. As a result, the generated font is *never* affected by any setting of the KEYMAP= graphics option.

CAUTION:
Fonts generated with the NOKEYMAP option are never affected by any setting of the KEYMAP= graphics option. △

By default, the NOKEYMAP option is *not* used; in which case, when you build a font, the current key map is applied to the values in the CHAR variable.

However, your current key map may not be symmetrical; that is, two or more input character codes may be mapped to the same output character. For example, if A is mapped to B, then both A and B map to B, but nothing maps to A. In this case, more than one code in your input data set can map to the same character in the resulting font. For example, if A and B are values of CHAR, both map to B. If this happens, a message that indicates the problem characters is displayed in the SAS log. To solve this problem, you can do one of the following:

- □ change the character code of one of the characters
- □ eliminate one of the characters
- □ use the NOKEYMAP option.

When you use the NOKEYMAP option, your font works correctly only if the end user's host or controller encoding is the same as the encoding used to create the input data set.

See also: the NOKEYMAP option on page 681 for Displaying Fonts and Chapter 18, "The GKEYMAP Procedure," on page 719

RESOL=1...4
R=1...4

controls the resolution of the fonts by specifying the number of bytes (1 through 4) for storing coordinates in the font. The GFONT procedure provides three resolution levels (RESOL=3 produces the same resolution level as RESOL=4). By default, RESOL=1.

The higher the number, the closer together the points that define the character can be spaced. A high value specifies a denser set of points for each character so that the characters approximate smooth curved lines at very large sizes. RESOL=2 works well for most applications; RESOL=3 or 4 may be too dense to be practical.

The table below shows the resolution number and the maximum number of distinct points that can be defined horizontally or vertically.

Resolution	Number of Distinct Points
2	32,766
3	2,147,483,646
4	2,147,483,646

Featured in: Example 2 on page 700

ROMHEX
HEX

specifies that hexadecimal values display automatically below the font characters when the GFONT procedure displays the font. If you use the ROMHEX option for a font that you create, you can later use the NOROMHEX option to suppress display of the hexadecimal values.

See also: the SHOWROMAN option on page 686, the ROMHEX option on page 682 for Displaying Fonts, and the NOROMHEX option on page 681

SHOWROMAN
SR

specifies that character codes display automatically below the font characters when the GFONT procedure displays the font. If you use the SHOWROMAN option for a font you create, you can later use the NOROMAN option to suppress display of the character codes.

See also: the ROMHEX option on page 686, the SHOWROMAN option on page 682 for Displaying Fonts, and the NOROMAN option on page 681

SPACEDATA=*space-data-set*
SPACE=*space-data-set*

specifies the SAS data set that contains font spacing information. When you use the SPACEDATA= option during font creation, the data contained in the space data set are applied to the font and stored with it. You cannot specify space adjustment for a double-byte character set that is created by using the option CODELEN=2.

See also: "The Space Data Set" on page 696

UNIFORM
U

specifies that characters are spaced uniformly rather than proportionately. Using the UNIFORM option is the same as specifying CHARSPACETYPE=UNIFORM.

See also: the CHARSPACETYPE= option on page 684 and the MWIDTH= option on page 685

Creating a Font

To create a font, you must create a data set that contains font information. Typically, you use a DATA step to create a SAS data set from which the GFONT procedure generates the font. The data set is referred to as the *font data set* and you can specify it with the DATA= argument.

To produce the font, invoke the GFONT procedure and specify the data set that contains the font information. In addition you can include options to modify the design and appearance of the font. For example, the following statement uses the data set FONTDATA to generate the font MYLOGO:

```
proc gfont data=fontdata name=mylogo;
```

For a demonstration of the font creation process, see Example 2 on page 700.

The GFONT procedure uses three types of data sets: the font data set, the kern data set, and the space data set. Each type of data set must contain certain variables and meet certain requirements. The following sections explain what each data set contains, how it is built, and what the requirements of the variables are.

The Font Data Set

The font data set consists of a series of observations that include the horizontal and vertical coordinate values and line segment numbers that the GFONT procedure uses to generate each character. In addition, each observation must include a character code that is associated with the font character and is used to specify the font character in a text string. The font data set also determines whether the font is stroked or polygon. A font data set that generates a polygon font produces an outline font by default. You can use the FILLED option with the same data set to generate a filled font.

The variables in the font data set must be assigned certain names and types. The table below summarizes the characteristics of the variables which are described further in "Font Data Set Variables" on page 688.

Table 16.1 Font Data Set Variables

Variable	Description	Type	Length	Valid Values	With Stroked Fonts	With Polygon Fonts
CHAR	the character code associated with the font character	character	1 or 2	keyboard characters or hexadecimal values	required	required
LP	the type of line segment being drawn, either a line or a polygon	character	1	L or P	optional	required
PTYPE	the type of data in the observation	character	1	V or C or W	optional	optional
SEGMENT	the number of the line segment or polygon being drawn	numeric		number	required	required
X	the horizontal coordinate	numeric		number	required	required
Y	the vertical coordinate	numeric		number	required	required

Font Data Set Variables

CHAR
provides a code for the character or figure that you are creating. CHAR is a character variable with a length of 1 or 2 and is required for all fonts.

CAUTION:
Using reserved or undefined hexadecimal codes as CHAR values may require the use of the NOKEYMAP option. △

The CHAR variable takes any character as its value, including characters that you can enter from your keyboard and hexadecimal values from ’00’x to ’FF’x. (If you use hexadecimal values as CHAR values, your font may not work correctly under a key map that is different from the one under which the font was created because positions that are not defined in one key map may be defined in another.)
When you specify the code character in a text string, the associated font character is drawn. For example, if you create a Roman alphabet font, typically the characters you specify for CHAR are keyboard characters that match the character in the font. All of the observations that build the letter A have a CHAR value of A. When you specify ’A’ in a text string this produces A in the output.
However, if you build a symbol font, the symbols may not have corresponding keyboard characters. In that case, you select a character or hexadecimal value to

represent each symbol in the font and assign it to CHAR. For example, in the Special font, the letter G is assigned as the code for the fleur-de-lis symbol. When you specify the code in a text string, the associated symbol displays.

If the CODELEN= option is set to 2, the values for CHAR represent two characters, such as AA, or a four-digit hexadecimal value, such as '00A5'x.

LP

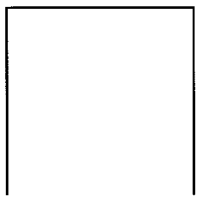
tells the GFONT procedure whether the coordinates of each segment form a line or a polygon. LP is a character variable with a length of 1. It is required for polygon fonts but optional for stroked fonts. You can assign the LP variable either of the following values:

L	lines
P	polygons.

Every group of line segments with an LP value of P is designated as a polygon; if the observations do not draw a completely closed figure, the program closes the figure automatically. For example, the following observations do not contain an LP variable. They produce a shape like the one in Figure 16.5 on page 689.

OBS	CHAR	SEG	X	Y
1	b	1	1	1
2	b	1	1	3
3	b	1	3	3
4	b	1	3	1

Figure 16.5 Using an LP Value of Line

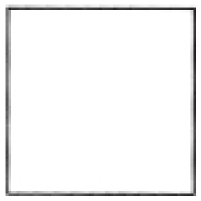


LP (continued)

An LP variable with a value of P for all observations added to the data set produces a complete box like the one in Figure 16.6 on page 690.

OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P

Figure 16.6 Using an LP Value of Polygon

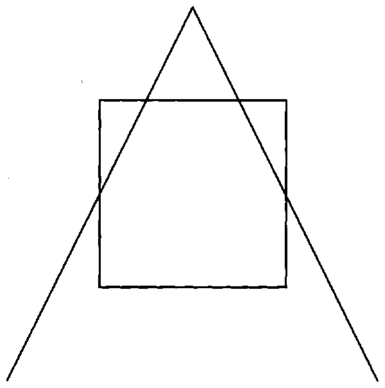


LP (continued)

The LP variable allows you to mix lines and polygons when you create characters in a font. For example, the following observations produce the single figure that is composed of a polygon and a line segment, as shown in Figure 16.7 on page 690:

OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P
5	b	2	0	0	L
6	b	2	2	4	L
7	b	2	4	0	L

Figure 16.7 Mixing LP Values of Line and Polygon



PTYPE

tells the GFONT procedure what type of data are in the observation. PTYPE is a character variable of length 1 that is optional for both stroked and polygon fonts. For each observation, the PTYPE variable assigns a characteristic to the point that is determined by the X and Y values. You can assign the PTYPE variable any of the following values:

V	normal point in the line segment
C	center of a circular arc joining two V points
W	width value for CHARSPACETYPE=DATA.

If the GFONT procedure encounters the sequence V-C-V in consecutive observations, it draws an arc that connects the two V points and has its center at the C point. If a circle cannot be centered at C and pass through both V points, the results are unpredictable. Arcs are limited to 106 degrees or less.

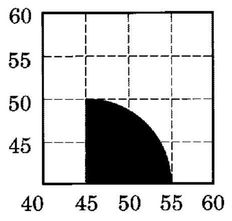
If you specify an observation with a PTYPE value of W, it must always be the first observation for a character. Instead of providing digitizing data to the procedure, the observation gives the minimum and maximum X values for the character. Note that in this case, the Y variable observation actually contains the maximum X value. Usually, these values include a little extra space for intercharacter spacing. Use a PTYPE of W only if you have specified CHARSPACETYPE=DATA; otherwise, the points are ignored. For more information on intercharacter spacing, see the description of the CHARSPACETYPE= option on page 684.

If you do not specify a PTYPE variable in the font data set, all points are assumed to be V-type points.

The following observations illustrate how the PTYPE variable is used to draw an arc similar to Figure 16.8 on page 692. (After the figure was generated, a grid was overlaid on it to show the location of the points.) A comment following each observation explains its function.

OBS	CHAR	SEG	X	Y	LP	PTYPE	Comment
1	a	1	40	60	P	W	define width of character as 20 font units, which is the number of units from left margin, 40, to right margin, 60
2	a	1	45	40	P	V	start line segment at position 45,40
3	a	1	45	50	P	V	draw a line to position 45,50, which is start point of arc
4	a	1	45	40	P	C	draw an arc whose center is at 45,40
5	a	1	55	40	P	V	finish drawing the arc at 55,40

Figure 16.8 Using the PTYPE Variable to Create an Arc



Note the following:

- □ Three observations are required to draw the arc: observation 3 and observation 5 denote the start point and endpoint of the arc, respectively, and observation 4 locates the center of the arc.
- □ The figure is closed because the line segments have an LP value of P (polygon).
- □ The font that contains the figure of the arc was generated with a PROC GFONT statement like the following:

  ```
  proc gfont data=arc name=arcfig charspacetype=data filled ;
  ```

  Note that the GFONT procedure uses the CHARSPACETYPE= option with a value of DATA to specify that the first observation sets the width of the character. The FILLED option fills the area of the arc.

SEGMENT

numbers the line segments that compose a character or symbol. SEGMENT is a numeric variable that is required for both polygon and stroked fonts. All the observations for a given line segment have the same segment number. The segment number changes when a new line segment starts.

When the GFONT procedure draws a stroked character with more than one line segment (for example, the letter E), or a polygon character with a hole (for example, the letter A), it needs to know when one line stops and where the next line begins. There are two ways to do this, as follows:

1. Change the segment number when a new line segment starts. If the value of LP is L (line), a change in segment numbers tells the GFONT procedure not to connect the last point in line segment 1 and the first point in line segment 2. If the value of LP is P (polygon), a change in segment numbers causes both of the following:
   - □ The last point in line segment 1 is joined to the first point in line segment 1, thus closing the polygon.
   - □ The program starts a new polygon. If the value of CHAR has not changed, the new polygon is part of the same character.

   Use this method for characters that are composed of two polygons, such as a question mark (?). If you draw a polygon with a hole in it, such as the letter A, use the second method.
2. Keep the same segment number for all lines, but insert an observation with missing values for X and Y between the observation that marks the end of the first line and the observation that begins the next line. For example, if you are drawing the letter O, insert an observation with a missing value between the line that draws the outer circle and the beginning of the line that draws the inner circle.

The first method is preferred, unless you are creating a polygon character with a hole in it. In this case, you should separate the lines with a missing value and keep the same segment numbers. (Note that if you use separate line segments when you create a polygon with a hole, the results may be unpredictable.) For example, observations such as the following from a data set called BOXES were used to draw the hollow square in Figure 16.9 on page 693. The data points that form the figure are laid out on a grid shown next to the square.

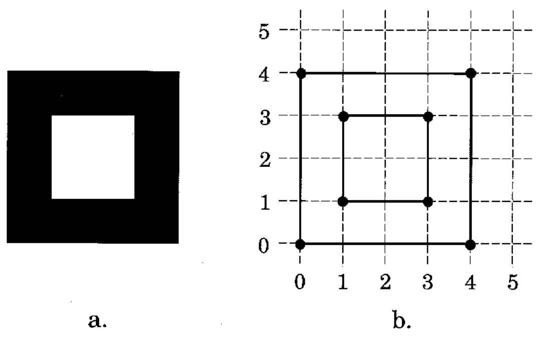
OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P
5	b	1	-	-	P
6	b	1	0	0	P
7	b	1	0	4	P
8	b	1	4	4	P
9	b	1	4	0	P

Note that observation 5, which has missing values for X and Y, separates the observations that draw the inner box from those that draw the outer box and that the segment number is the same for all the observations. Figure 16.9 on page 693 was generated with a GFONT statement like the following:

```
proc gfont data=boxes name=boxes filled;
```

Note that the FILLED option is included and that only the space between the two squares is filled.

Figure 16.9 Drawing Nested Polygons



X Y

specify the horizontal and vertical coordinates of the points for each character. These variables must be numeric, and they must be named X and Y for the horizontal and vertical coordinates, respectively. Their values describe the position of the points on the character. These values can be in any range that you choose, but both variables must describe the character in the same scale or font units. In other words, 10 horizontal units must be the same distance as 10 vertical units. You should define vertical coordinates for all characters on the same baseline.

Note: When you specify PTYPE=W, both X and Y contain horizontal coordinate values. △

Creating a Font Data Set

You can create a font data set by digitizing the shape of the characters or figures either manually or with special digitizing equipment. To create a font data set by digitizing the characters manually, follow these steps:

1. Determine the coordinate points for each line segment by drawing the characters on a grid.
2. Lay out the observations for each character. Each observation describes a move from one point to another along a line segment. For each line segment, enter the coordinate points in the order in which they are drawn. For a stroked font, when you start a new line segment, change the segment number. For a polygon font, when you start a new polygon, change the line segment number.

   If the polygon has a hole in it, as in the letter O, keep the line segment number and separate the lines with a missing value. Use the same value for CHAR for all of the observations that describe one character.
3. Create a SAS data set that contains the variables CHAR, SEGMENT, X, and Y, and read in the data for each observation. Include the variables LP and PTYPE if necessary.
4. Sort the data set by CHAR and SEGMENT.
5. Assign the font data set with the DATA= argument.

This process is illustrated in Example 2 on page 700.

The Kern Data Set

The kern data set consists of observations that specify how much space to add or remove between any two characters when they appear in combination. This process, called *kerning*, increases or decreases space between the characters. Kerning usually is applied to certain pairs of characters that, because of their shape, have too much space between them. Reducing the space between characters may allow part of one character to extend over the body of the next. Examples of some combinations that should be kerned are AT, AV, AW, TA, VA, and WA.

You can apply kerning to the intercharacter spacing that you specify with the CHARSPACETYPE= option (except for uniform fonts). You can refine the kerning of your characters as little or as much as you like. You assign the kern data set with the KERNDATA= option.

Kern Data Set Variables

The kern data set must contain these variables:

CHAR1
: specifies the first character in the pair to be kerned. CHAR1 is a character variable with a length of 1.

CHAR2
: specifies the second character in the pair to be kerned. CHAR2 is a character variable with a length of 1.

XADJ
: specifies the amount of space to add or remove between the two characters. XADJ is a numeric variable that uses the same font units as the font data set. The value of XADJ specifies the horizontal adjustment to be applied to CHAR2 whenever CHAR1 is followed immediately by CHAR2. Negative numbers decrease the spacing, and positive numbers increase the spacing.

Creating a Kern Data Set

Each observation in a kern data set names the pair of characters to be kerned and the amount of space to be added or deleted between them. To create a kern data set, follow these steps:

1 Select the pairs of characters to be kerned, and specify the space adjustment (in font units) for each pair as a positive number (more space) or negative number (less space).

2 Create a SAS data set that contains the variables CHAR1, CHAR2, and XADJ; produce one observation for each pair of characters and the corresponding space adjustment.

```
data kern1;
   input char1 $ char2 $ xadj;
   datalines;
A T -4
D A -3
T A -4
;
```

3 Assign the kern data set with the KERNDATA= option.

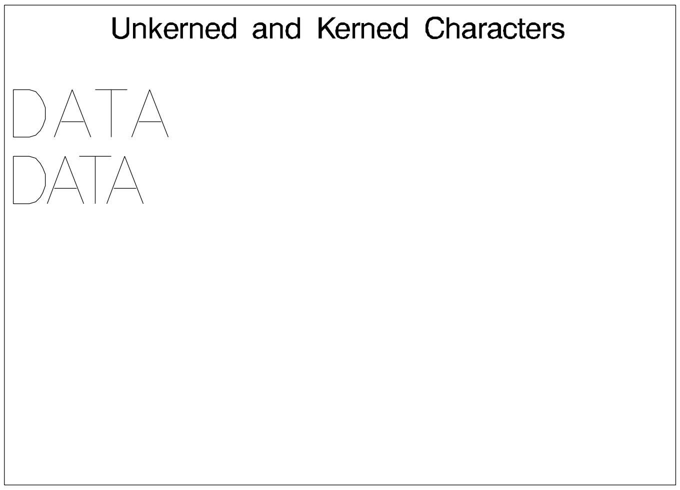
```
proc gfont data=fontdata
           name=font2
           charspacetype=data
           kerndata=kern1
           nodisplay;
run;
```

Figure 16.10 on page 696 illustrates how you can use the KERNDATA= option to create a font in which the space between specified pairs of letters is reduced. The characters A, D, and T are shown as the word DATA. The first line uses the unkerned font, FONT1, and the second line uses the kerned font, FONT2. Note that the characters in FONT2 are spaced more closely than the characters in FONT1.

The following title statements specify the kerned and unkerned fonts and are used with the GSLIDE procedure to produce Figure 16.10 on page 696:

```
title2 lspace=6 f=font1 h=10 j=l 'DATA';
title3 lspace=4 f=font2 h=10 j=l 'DATA';
```

Figure 16.10 Comparison of Kerned and Unkerned Text



The Space Data Set

As the height (point size) of a font increases, less space is required between letters in relation to their height. If the point size decreases, more space may be needed. The space data set tells the GFONT procedure how much to increase or decrease the intercharacter spacing for a given point size. Like kerning, spacing is added to or subtracted from the intercharacter spacing that is specified by the CHARSPACETYPE= option. However, kerning applies the adjustment to specified pairs of characters, while spacing is applied uniformly to all characters.

Values that are specified in the space data set are added to the normal intercharacter spacing and any kerning data. Normal intercharacter spacing is determined by the CHARSPACETYPE= option.

Space Data Set Variables

The space data set must contain these variables:

SIZE
: specifies the point size of the font. SIZE is a numeric variable.

ADJ
: specifies the spacing adjustment for the point size in hundredths (1/100) of a point. (A point is equal to 1/72 of an inch.) ADJ is a numeric variable. Positive values for the ADJ variable increase the spacing between characters; negative values reduce the space.

Creating a Space Data Set

Each observation in a space data set specifies a point size (SIZE) and the amount of space (ADJ) to be added or subtracted between characters when a font of that point size is requested. When you specify a point size that is not in the space data set, the adjustment for the next smaller size is used. To create a space data set, follow these steps:

1 Determine the amount of adjustment that is required for typical point sizes; positive numbers increase spacing, and negative numbers decrease spacing.

2 Create a SAS data set that contains the variables SIZE and ADJ; produce one observation for each point size and corresponding space adjustment.

```
data space1;
   input size adj;
   datalines;
 6   40
12    0
18  -40
24  -90
30 -150
36 -300
42 -620
;
```

3 Assign the space data set with the SPACEDATA= option.

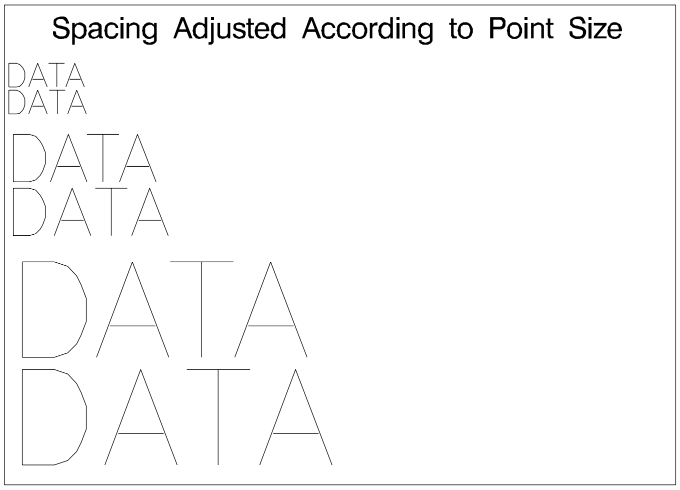
```
proc gfont data=reflib.fontdata
           name=font3
           charspacetype=data
           spacedata=space1
           nodisplay;
run;
```

Figure 16.11 on page 698 illustrates how to use the SPACEDATA= option to generate a font in which intercharacter spacing is adjusted according to the height of the characters. The characters A, D, and T are shown as the word DATA. Each pair of lines displays the word DATA and at the same size uses first the font with spacing adjustment (FONT3) and then the original font (FONT1). Note that as the size of the characters increases, the space between them decreases.

The following title statements are used with the GSLIDE procedure to produce Figure 16.11 on page 698:

```
title2;
title3 f=font3 h=.25in j=l 'DATA'; /* 18 points */
title4 f=font1 h=.25in j=l 'DATA';
title5;
title6 f=font3 h=.50in j=l 'DATA'; /* 36 points */
title7 f=font1 h=.50in j=l 'DATA';
title8;
title9 f=font3 h=1.0in j=l 'DATA'; /* 72 points */
title10 f=font1 h=1.0in j=l 'DATA';
```

Figure 16.11 Comparison of Text with and without Spacing Adjustments



Examples

The following examples illustrate major features of the GFONT procedure.

Example 1: Displaying Fonts and Character Codes

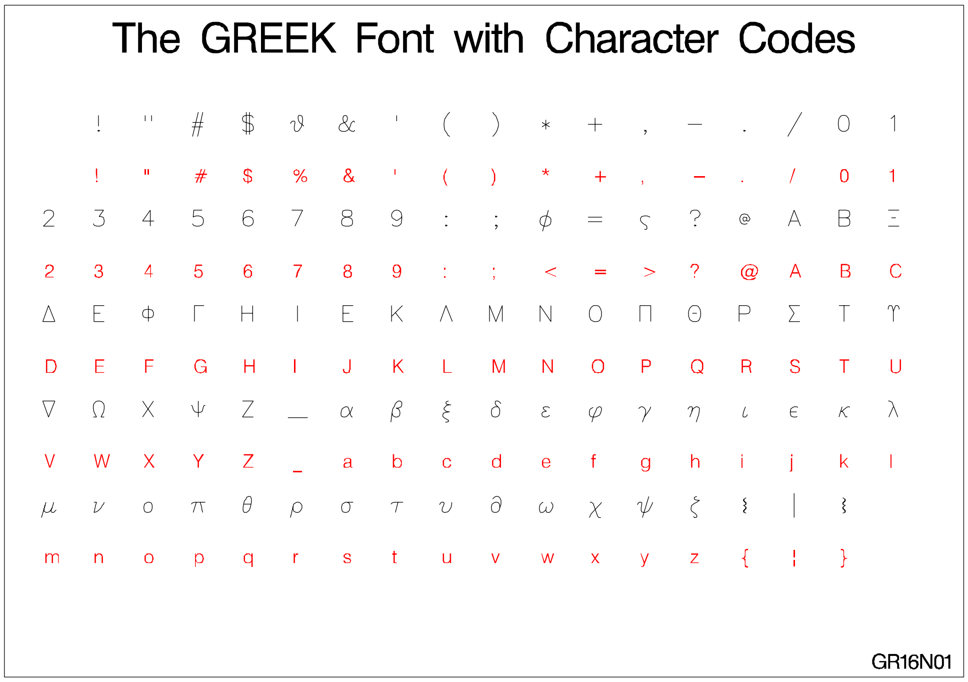
Procedure features:

GFONT statement options:

- HEIGHT=
- NOBUILD
- ROMCOL=
- ROMFONT=
- ROMHT=
- SHOWROMAN

Sample library member: GR16N01

Figure 16.12 Display of the Greek Font with Character Codes (GR16N01)


The GREEK Font with Character Codes
GR16N01

This example illustrates the SHOWROMAN option, which displays the character codes that are associated with the font characters that are being displayed. A display such as this one shows which keyboard character you enter to produce the Greek character you want. In addition, this example shows how to modify the appearance of both the font characters and the character codes when they are displayed.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Define title and footnote.

```
title 'The GREEK Font with Character Codes';
footnote j=r 'GR16N01 ';
```

Display the GREEK font with character codes. NOBUILD indicates that the font specified in the NAME= argument is an existing font. HEIGHT= specifies the height of the Greek characters. ROMCOL=, ROMFONT=, and ROMHT= assign the color, type style, and height of the character codes. SHOWROMAN displays the character codes.

```
proc gfont name=greek
           nobuild
           height=3.7
           romcol=red
           romfont=swissl
           romht=2.7
```

```
             showroman;
run;
quit;
```

Example 2: Creating Figures for a Symbol Font

Procedure features:

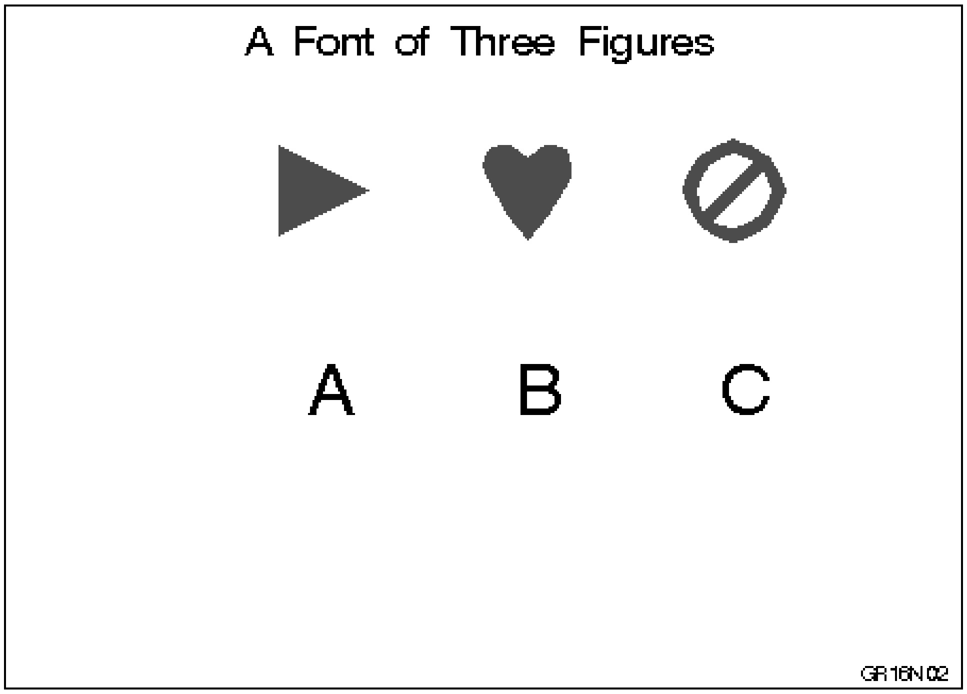
GFONT statement options:

CTEXT=
DATA=
FILLED
NAME=
RESOL=

Other features:

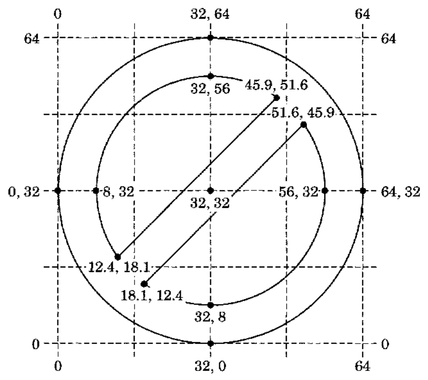
LIBNAME statement

Sample library member: GR16N02



This example shows how to create three simple figures for a symbol font. Each figure is laid out on a grid that is 64 font units square. The third figure is a circle with a slash through it. Figure 16.13 on page 701 shows the figure and some of its coordinate points laid out on a grid.

Figure 16.13 Diagram of Circle with Slash Figure



Assign the librefs and set the graphics environment. REFLIB is the permanent library that is used to store the data set. The second LIBNAME statement associates the libref GFONT0 with the SAS data library in which the font catalog is stored.

```
libname reflib 'SAS-data-library';
libname gfont0 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Create the font data set REFLIB.FIGURES for a triangle, a heart, and a circle with slash. The first figure, a right-pointing triangle that is assigned the character code A, is a polygon drawn with three straight lines.

```
data reflib.figures;
   input char $ ptype $ x y segment lp $;
   datalines;
A    W     0    64    0    P   /* triangle pointing right */
A    V     4     4    1    P
A    V    60    32    1    P
A    V     4    60    1    P
A    V     4     4    1    P
```

The second figure, a heart that is assigned the character code B, uses the PTYPE variable combination V-C-V to draw the arcs that make up the top of the heart. Each side requires two arcs. Since the arcs are continuous, the observation that marks the end of one arc is also the beginning of the next arc. The heart drawing begins at the bottom point and continues counterclockwise.

```
B    W     0    64    0    P   /* heart */
B    V    32     2    1    P
B    V    44    17    1    P
B    V    58    40    1    P
B    C    46    47    1    P
```

```
B    V   56    58   1    P
B    C   46    47   1    P
B    V   32    52   1    P
B    C   18    47   1    P
B    V    8    58   1    P
B    C   18    47   1    P
B    V    6    40   1    P
B    V   20    17   1    P
B    V   32     2   1    P
```

The third figure, a circle with a slash through it that is assigned the character code C, is composed of three polygons: a circle and two empty arcs. An observation with missing values separates the observations defining each of the three polygons. The outer circle is defined by the first group of observations. The empty arcs are drawn with three continuous arcs using the PTYPE variable pattern V-C-V-C-V-C-V. The straight line that closes the arc is drawn automatically by the GFONT procedure in order to complete the polygon. Because all the polygons are part of one character, the continuous space they define is filled.

```
C    W    0    64   0    P  /* circle with slash */
C    V   32    64   1    P
C    C   32    32   1    P
C    V   64    32   1    P
C    C   32    32   1    P
C    V   32     0   1    P
C    C   32    32   1    P
C    V    0    32   1    P
C    C   32    32   1    P
C    V   32    64   1    P
C    V    .     .   1    P
C    V   12.4  18.1 1    P
C    C   32    32   1    P
C    V    8    32   1    P
C    C   32    32   1    P
C    V   32    56   1    P
C    C   32    32   1    P
C    V   45.9  51.6 1    P
C    V    .     .   1    P
C    V   51.6  45.9 1    P
C    C   32    32   1    P
C    V   56    32   1    P
C    C   32    32   1    P
C    V   32     8   1    P
C    C   32    32   1    P
C    V   18.1  12.4 1    P
;
```

Define the title and footnote.

```
title 'A Font of Three Figures';
footnote j=r 'GR16N02 ';
```

Generate and display the font FIGURES. The DATA= argument names the input data set that is used to generate the font. The NAME= argument names the font that the procedure generates and automatically stores it in the GFONT0 catalog. (Note that you do not need to specify GFONT0.) FILLED specifies a filled polygon font. CTEXT= specifies the color of the figures in the font display. The color specification is not stored with the font. RESOL= is set to 2 to improve the resolution of the lines. By default, the newly generated font is displayed (the NODISPLAY option is not used).

```
proc gfont data=reflib.figures
           name=figures
           filled
           height=.75in
           ctext=red
           showroman
           romht=.5in
           resol=2;
run;
quit;
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/GRAPH® Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/GRAPH® Software: Reference, Version 8

ISBN 1-58025-525-6

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999