Chapter 3 -- Shapes, Fonts, and PostScript Support

Shape Definition Files

AutoCAD font and shape files (.*shx*) are generated (compiled) from shape definition files (.*shp*). You can create or modify shape definition files with a text editor or word processor that saves files in ASCII format.

The syntax of the shape description for each shape (or character) is the same regardless of the final use (shape or font) for that shape description. If a shape definition file is to be used as a font file, the first entry in the file describes the font itself rather than a shape within the file. If this initial entry describes a shape, the file is used as a shape file.

AutoCAD comes with two sample shape files: *pc.shx* and *es.shx*. One is for printed circuit layout and the other is for electronic schematics. Examining the contents of these files and modifying their shape descriptions can help you master the definition of AutoCAD shapes.

Topics in this section:

Shape Definition Files

Shape Descriptions

Each line in a shape definition file can contain up to 128 characters. Longer lines cannot be compiled. AutoCAD ignores blank lines and text to the right of a semicolon. The semicolon provides the ability to embed comments in shape definition files.

Each shape description has a header line of the following form and is followed by one or more lines containing specification bytes, separated by commas and terminated by a 0.

```
*shapenumber,defbytes,shapename
specbyte1,specbytec2,specbytec3,...,0
```

The following describes the fields of a shape description:

shapenumber

A number, unique to the file, between 1 and 258, and preceded by an asterisk (*). Every shape in a shape file must have a number (numbers 256, 257, and 258 are for the symbolic identifiers Degree_Sign, Plus_Or_Minus_Sign, and Diameter_Symbol). Text fonts (files containing shape definitions for each character) require specific numbers corresponding to the value of each character in the ASCII code; other shapes can be assigned any numbers.

defbytes

The number of data bytes (*specbytes*) required to describe the shape, including the terminating zero. The limit is 2,000 bytes per shape.

shapename

The shape name. Shape names must be upper case to be recognized. Names with lowercase characters are ignored and are usually used to label font shape definitions.

specbyte

A shape specification byte. Each specification byte is a code that defines either a vector length and direction or one of a number of special codes. A specification byte can be expressed in the shape definition file as either a decimal or hexadecimal value. This section uses both decimal and hexadecimal specification byte values for its examples (as do many of the shape definition files). If the first character of a specification byte is a 0 (zero), the following two characters are interpreted as hexadecimal values.

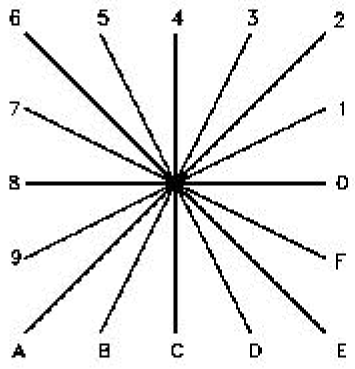
Topics in this section:

Shape Definition Files

Shape Descriptions

Vector Length and Direction Code

A simple shape specification byte contains vector length and direction encoded into one specification byte (one `specbyte` field). Each vector length and direction code is a three character string. The first character must be a 0, which indicates to AutoCAD that the next two characters are interpreted as hexadecimal values. The second character specifies the length of the vector in units. Valid hexadecimal values range from 1 (one unit long) through F (15 units long). The third character specifies the direction of the vector. The following figure illustrates the direction codes.



Vector direction codes

All the vectors in the preceding figure were drawn with the same length specification. Diagonal vectors stretch to match the *X* or *Y* displacement of the closest orthogonal vector. This is similar to the action of the snap grid in AutoCAD.

The following example constructs a shape named DBOX with an arbitrarily assigned shape number of 230.

```
*230,6,DBOX
014,010,01C,018,012,0
```

The preceding sequence of specification bytes defines a box one unit high by one unit wide, with a diagonal line running from the lower-left corner to the upper-right corner. After saving the files as *dbox.shp*, use the COMPILE command to generate the *dbox.shx* file. Use the LOAD command to load the shape file containing this definition, and then use the SHAPE command as follows:

Command: **shape**

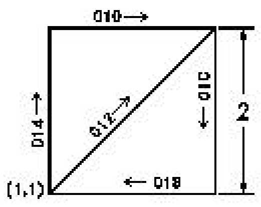
Name (or ?): **dbox**

Starting point: **1,1**

Height <*current*>: **2**

Rotation angle <*current*>: **0**

The resulting shape is shown in the following illustration.



Shape Definition Files

Shape Descriptions

Special Codes

In addition to defining vectors, a specification byte can use the following special codes to create additional forms and specify certain actions. To use a special code, the second character of the three character string (the vector length specification) must be 0, or you can specify only the code number (for example, 008 and 8 are both valid specifications).

Specification byte codes

Code	Description
000	End of shape definition
001	Activate Draw mode (pen down)
002	Deactivate Draw mode (pen up)
003	Divide vector lengths by next byte
004	Multiply vector lengths by next byte
005	Push current location onto stack
006	Pop current location from stack
007	Draw subshape number given by next byte
008	*X-Y* displacement given by next two bytes
009	Multiple *X-Y* displacements, terminated (0,0)
00A	Octant arc defined by next two bytes
00B	Fractional arc defined by next five bytes
00C	Arc defined by *X-Y* displacement and bulge
00D	Multiple bulge-specified arcs
00E	Process next command only if vertical text

Code 0: End of Shape

Code 0 marks the end of the shape definition.

Codes 1 and 2: Draw Mode Control

Codes 1 and 2 control Draw mode. Draw is activated at the start of each shape. When Draw mode is turned on, the vectors cause lines to be drawn. When Draw mode is turned off, the vectors move to a new location without drawing.

Codes 3 and 4: Size Control

Codes 3 and 4 control the relative size of each vector. The height specified with the SHAPE command is initially considered the length of a single orthogonal vector (directions 0, 4, 8, or C). Codes 3 and 4 are followed by a specification byte containing an integer scale factor (1 through 255). If you want the shape height to specify the size of the entire shape, and you use 10 vector lengths to draw it, you can use 3,10 to scale the height specification. The scale factor is cumulative within a shape; that is, multiplying by 2 and again by 6 results in a scale factor of 12. Usually you will want to reverse the effect of your scale factors at the end of the shape, especially for subshapes and text font shapes. AutoCAD does not reset the scale factor for you.

Codes 5 and 6: Location Save/Restore

Codes 5 and 6 push (save) and pop (restore) the current coordinate position while drawing a shape so that you can return to it from a later point in the shape. You must pop everything you push. The position stack is only four locations deep. If the stack overflows because of too many pushes or too many missing pops, the following message is displayed when the shape is drawn.

    Position stack overflow in shape *nnn*

Similarly, if you try to pop more locations than have been pushed onto the stack, the following message is displayed when the shape is drawn.

    Position stack underflow in shape *nnn*

Code 7: Subshape

Code 7 is a subshape reference. The specification byte following code 7 is a shape number between 1 and 255. The shape with that number (in the same shape file) is drawn at this time. Note that Draw mode is not reset for the new shape. When the subshape is complete, drawing the current shape resumes.

Codes 8 and 9: *X-Y* Displacements

Normal vector specification bytes draw only in the 16 predefined directions, and the longest length is 15. These restrictions help make shape definitions efficient but are sometimes limiting. With codes 8 and 9 you can draw nonstandard vectors using *X, Y* displacements. Code 8 must be followed by two specification bytes in the format:

`8,`*X-displacement*`,`*Y-displacement*

The *X*, *Y* displacements can range from -128 to +127. A leading + is optional, and you can use parentheses to improve readability. The following example results in a vector that draws (or moves) 10 units to the left and three units up.

```
8,(-10,3)
```

Following the two displacement specification bytes, the shape returns to Normal Vector mode.

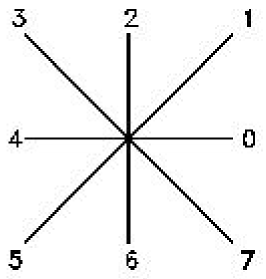
You can use code 9 to draw a sequence of nonstandard vectors. This code can be followed by any number of *X, Y* displacement pairs. It is terminated by a (0,0) pair. The following example draws three nonstandard vectors and returns to Normal Vector mode.

```
9,(3,1),(3,2),(2,-3),(0,0)
```

Note that you must terminate the sequence of *X, Y* displacement pairs with a (0,0) pair in order for AutoCAD to recognize any Normal Vectors or special codes that follow.

Code 00A: Octant Arc

Special code 00A (or 10) uses the next two specification bytes to define an arc. This is called an *octant arc* because it spans one or more 45-degree *octants*, starting and ending on an octant boundary. Octants are numbered counterclockwise from the 3 o'clock position, as shown in the following illustration.



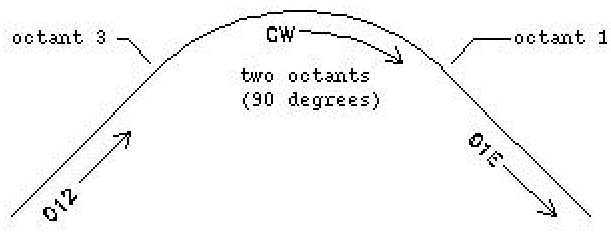
The arc specification is

`10,`*radius*`,(-)0`*SC*

The radius can be any value from 1 through 255. The second specification byte indicates the direction of the arc (counterclockwise if positive, and clockwise if negative), its starting octant (S--a value from 0 through 7), and the number of octants it spans (C--a value from 0 through 7, in which 0 equals eight octants, or a full circle). You can use parentheses to improve readability. For example, consider the following fragment of a shape definition:

```
...012,10,(1,-032),01E,...
```

This code draws a one-unit vector up and to the right, a clockwise arc from octant 3 (with a radius of one unit for two octants), and then a one-unit vector down and to the right, as shown in the following illustration.



Code 00B: Fractional Arc

Special code 00B (11) draws an arc that doesn't necessarily start and end on an octant boundary. The definition uses five specification bytes.

`11,`*start_offset*`,`*end_offset*`,`*high_radius*`,`*radius*`,(-)0`*SC*

The *`start_offset`* and *`end_offset`* represent how far from an octant boundary the arc begins or ends. The *`high_radius`* represents the most significant eight bits of the radius; it will be 0 unless the *`radius`* is greater than 255 units. Multiply the *`high_radius`* value by 256 and add that value to the *`radius`* value to generate an arc radius greater than 255. The *`radius`* and ending specification byte are the same as for the octant arc specification (code 00A, described previously).

You determine the `start offset` by calculating the difference in degrees between the starting octant's boundary (a multiple of 45 degrees) and the start of the arc. Then you multiply this difference by 256 and divide by 45. If the arc starts on an octant boundary, its `start offset` is 0.

The `end offset` is calculated in a similar fashion, but you use the number of degrees from the last octant boundary crossed to the end of the arc. If the arc ends on an octant boundary, its `end offset` is 0.

For example, a fractional arc from 55 degrees to 95 degrees with a 3 unit radius would be coded as follows:

```
11,(56,28,0,3,012)
```

Here is the explanation:

```
start_offset     = 56 because ((55 - 45) * 256 / 45) = 56
end_offset       = 28 because ((95 - 90) * 256 / 45) = 28
high_radius      = 0  because (radius < 255)
radius           = 3
starting octant  = 1  because arc starts in the 45 degree octant
ending octant    = 2  because arc ends in the 90 degree octant
```

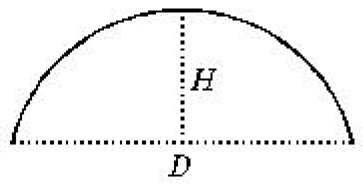
Codes 00C and 00D: Bulge-Specified Arcs

Special codes 00C and 00D (12 and 13) provide another mechanism for including arc segments in shape descriptions. They are similar to codes 8 and 9 in that you can use them to specify *X, Y* displacements. However, codes 00C and 00D let you draw arcs by applying a *bulge factor* to the displacement vector. Code 00C draws one arc segment, while code 00D draws multiple arc segments *(polyarcs)* until it is terminated by a (0,0) displacement.

Code 00C must be followed by three bytes describing the arc:

`0C,`*X-displacement*`,`*Y-displacement*`,`*Bulge*

Both the *X* and *Y* displacement and the bulge, which specifies the curvature of the arc, can range from -127 to +127. If the line segment specified by the displacement has length *D*, and the perpendicular distance from the midpoint of that segment has height *H*, the magnitude of the bulge is ((2 * *H* / *D*) * 127). The sign is negative if the arc from the current location to the new location is clockwise.



A semicircle has bulge 127 (or -127) and is the greatest arc that can be represented as a single-arc segment using these codes (use two consecutive arc segments for larger arcs). A bulge specification of 0 is valid and represents a straight-line segment. Note, however, that using code 8 for a straight-line segment saves a byte in the shape description.

The polyarc code (00D, or 13) is followed by 0 or by more arc segment triples, and is terminated by a (0,0) displacement. (Note that no bulge is specified after the final displacement.) For example, the letter *S* might be defined by the following sequence:

```
13,(0,5,127),(0,5,-127),(0,0)
```

Zero bulge segments are useful within polyarcs to represent straight segments; they are more efficient than terminating the polyarc, inserting one straight segment, and then starting another polyarc.

The number -128 cannot be used in arc segment and polyarc definitions.

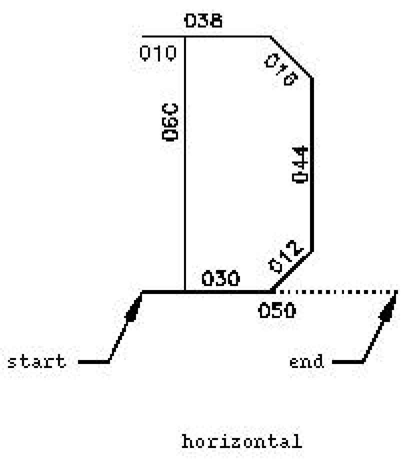
Code 00E: Flag Vertical Text Command

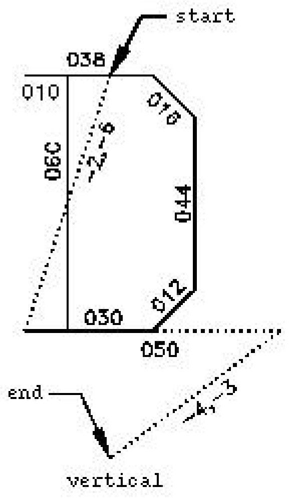
Special code 00E (14) is used only in dual-orientation text font descriptions, where the font is used in both horizontal and vertical orientations. When this special code is encountered in a character definition, the next code is either processed or skipped, depending on orientation. If the orientation is vertical, the next code is processed; if it is horizontal, the next code is skipped.

In horizontal text, the start point for each character is the left end of the baseline. In vertical text, the start point is assumed to be the top center of the character. At the end of each character, a pen-up segment is normally drawn to position to the next character's start point. For horizontal text, it is to the right; for vertical text, it is downward. The special 00E (14) code is used primarily to adjust for differences in start points and endpoints, permitting the same character shape definition to be used both horizontally and vertically. For instance, the following definition of an uppercase D could be used in either horizontal or vertical text.

```
*68,22,ucd
```

```
2,14,8,(-2, 6),1,030,012,044,016,038,2,010,1,06C,2,050,
14,8,(-4,-3),0
```





Shape Definition Files

Text Font Descriptions

AutoCAD is packaged with numerous text fonts. You can use the STYLE command to apply expansion, compression, or obliquing to any of these fonts, thereby tailoring the characters to your needs. You can draw text of any height, at any baseline angle, and with either horizontal or vertical orientation using these fonts.

If you intend to create your own text fonts, study the *txt.shp* file and the other fonts that are packaged with AutoCAD. They provide working examples of the topics discussed here. AutoCAD text fonts are files of shape definitions with shape numbers corresponding to the ASCII code for each character. For a list of the ASCII codes, see appendix A, "ASCII Codes."

Codes 1 through 31 are for control characters, only one of which is used in AutoCAD text fonts:

10 (LF)

The line feed (LF) must drop down one line without drawing. This is used for repeated TEXT commands, to place succeeding lines below the first one.

```
*10,5,lf
2,8,(0,-10),0
```

You can modify the spacing of lines by adjusting the downward movement specified by the LF shape definition.

Text fonts must include a special shape number 0 that conveys information about the font itself. The format has the following syntax:

```
*0,4,font-name
above,below,modes,0
```

where *above* specifies the number of vector lengths above the baseline that the uppercase letters extend, and *below* indicates how far the lowercase letters descend below the baseline. The baseline is similar in concept to the lines on writing paper. These values define the basic character size and are used as scale factors for the height specified in the TEXT command.

The *modes* byte should be 0 for a horizontally oriented font and 2 for a dual-orientation (horizontal or vertical) font. The special 00E (14) command code is honored only when *modes* is set to 2.

The standard fonts supplied with AutoCAD include a few additional characters required for the AutoCAD dimensioning feature.

%%d - Degree symbol (?)
%%p - Plus/minus tolerance symbol (?)
%%c - Circle diameter dimensioning symbol (?)

You can use these and the %%*nnn* control sequences, as described under "TEXT" in the *Command Reference*.

Note AutoCAD draws text characters by their ASCII codes (shape numbers) and not by name. To save memory, specify the shape name portion of each text shape definition in lower case as shown in the following example. (Lowercase names are not saved in memory.)

```
*65,11,uca
024,043,04d,02c,2,047,1,040,2,02e,0
```

Because the shape name `uca` contains lowercase letters, AutoCAD doesn't save the name in memory. However, you can use the name for reference when editing the font definition file. In this example, `uca` stands for uppercase A.

Chapter 3 -- Shapes, Fonts, and PostScript Support

Shape Definition Files

Big Font Descriptions

Some languages, such as Japanese, use text fonts with thousands of non-ASCII characters. In order for drawings to contain such text, AutoCAD supports a special form of shape definition file called a *big font* file.

Topics in this section:

Defining a Big Font

A font with hundreds or thousands of characters must be handled differently from a font containing the ASCII set of up to 256 characters. In addition to using more complicated techniques for searching the file, AutoCAD needs a way to represent characters with two-byte codes as well as one-byte codes. Both situations are addressed by the use of special codes at the beginning of a big font file.

The first line of a big font shape definition file must be

```
*BIGFONT nchars,nranges,b1,e1,b2,e2,...
```

where `nchars` is the approximate number of character definitions in this set; if it is off by more than about 10 percent, either speed or file size suffers. You can use the rest of the line to name special character codes (escape codes) that signify the start of a two-byte code. For example, on Japanese computers, Kanji characters start with hexadecimal codes in the range 90-AF or E0-FF. When the operating system sees one of these codes, it reads the next byte and combines the two bytes into a code for one Kanji character. In the `*BIGFONT` line, `nranges` tells how many contiguous ranges of numbers are used as escape codes; `b1`, `e1`, `b2`, `e2`, and so on, define the beginning and ending codes in each range. Therefore, the header for a Japanese big font file might look like this:

```
*BIGFONT 4000,2,090,0AF,0E0,0FF
```

After the `*BIGFONT` line, the font definition is just like a regular AutoCAD text font, except that character codes (shape numbers) can have values up to 65535.

Defining an Extended Big Font File

To reduce the size of composite Kanji characters, you can define an extended big font file. Extended big fonts use the subshape code, followed immediately by a 0.

The first line of an extended big font file is the same as the regular big font file. This is the format for the remaining lines of the file:

```
*0,5,font-name
character-height, 0, modes, character-width,0
   .
   .
   .
*shape-number,defbytes,shape-name
   .
code,0,primitive#,basepoint-x,basepoint-y,width,height,
   .
   .
code,0,primitive#,basepoint-x,basepoint-y,width,height,
   .
terminator
```

character- height

Used along with character width to indicate the number of units that define the font characters.

character- width

Used along with character height to indicate the number of units that define the font characters. The `character-height` and `character-width` values are used to scale the primitives of the font. In this context, primitives are the points, lines, polygons, or character strings of the font geometrically oriented in two-dimensional space. A Kanji character consists of several primitives used repeatedly in different scales and combinations.

modes

Same as for regular text fonts.

shape-number

Character code.

defbytes

Byte size. It is always 2 bytes, consisting of a hexadecimal or a combination of decimal and hexadecimal codes.

shape-name

Character name.

code

Shape description special code. It is always 7 so that it can use the subshape feature.

primitive#

Reference to the subshape number. It is always 2.

basepoint-x

X origin of the primitive.

basepoint-y

Y origin of the primitive.

width

Scale of the width of the primitive.

height

Scale of the height of the primitive.

terminator

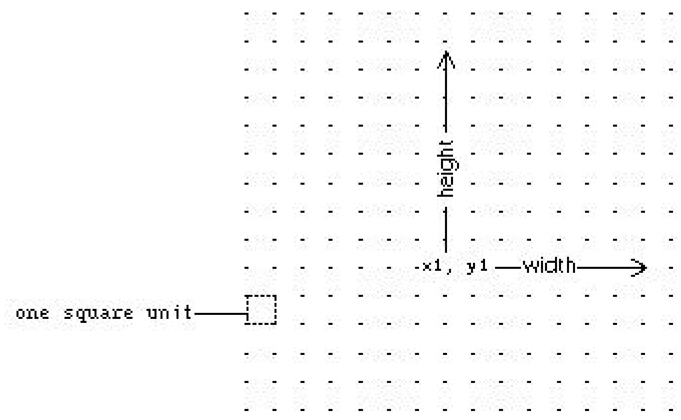
End-of-file indicator for the shape definition; It is always 0.

To arrive at the scale factor, AutoCAD scales down the primitive to a square unit and then multiplies it by the height and width to get the shape of the character. Character codes (shape numbers) in the big font shape definition file can have values up to 65535. The following table describes the fields of the extended big font file.

Fields of the extended big font file

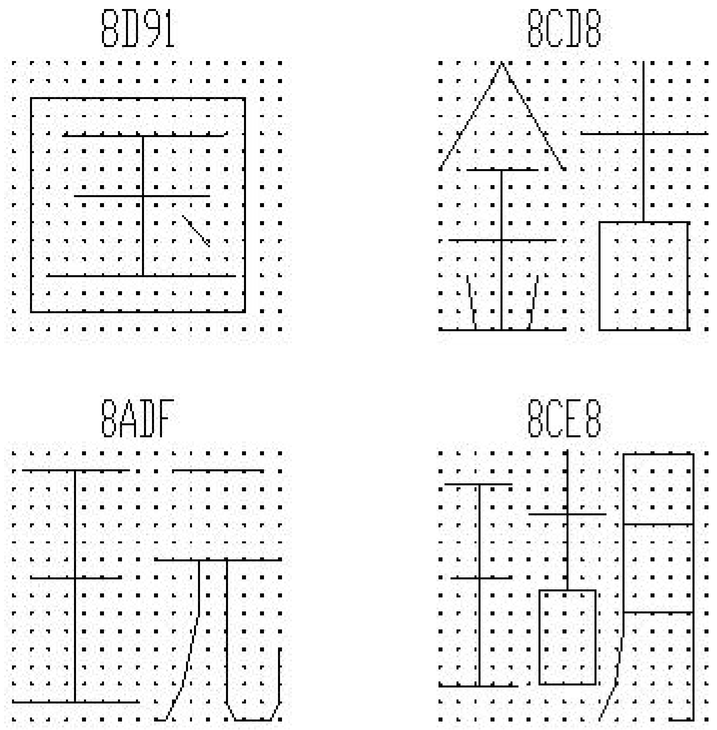
Variable	Value	Byte size	Description
`shape-number`	xxxx	2 bytes	Character code
`code`	7,0	2 bytes	Extended font definition
`primitive#`	xxxx	2 bytes	Refer to subshape number
`basepoint-x`		1 byte	Primitive *X* origin
`basepoint-y`		1 byte	Primitive *Y* origin
`width`		1 byte	Scale of primitive width
`height`		1 byte	Scale of primitive height
`terminator`	0	1 byte	End of shape definition

The following figure is an example of a 16x16 dot matrix that you could use to design an extended big font, such as a Kanji character. In the example, the distance between each dot is one unit. The callout points to a square unit.



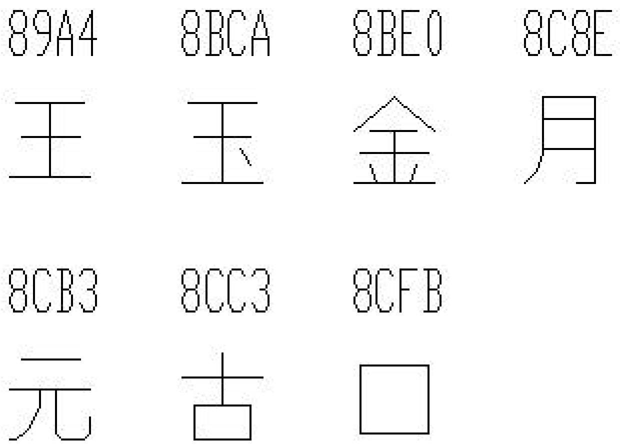
A square matrix for a Kanji character

The following figure shows examples of Kanji characters. Each character occupies an M x N matrix, (matrices don't have to be square) similar to the one shown in the previous figure.



Examples of Kanji characters

The following figure shows Kanji primitives.



Examples of Kanji primitives

Note Not all fonts are defined in a square matrix; some are defined in rectangular matrices.

The following is an example of a shape definition file for an extended big font.

```
*BIGFONT 50,1,080,09e
*0,5,Extended Font
15,0,2,15,0
*08D91,31,unspecified
2,0e,8,-7,-15,
7,0,08cfb,0,0,16,16,7,0,08bca,2,3,12,9,
2,8,18,0,2,0e,8,-11,-3,0
*08CD8,31,unspecified
2,0e,8,-7,-15,
7,0,08be0,0,0,8,16,7,0,08cc3,8,0,8,16,
2,8,18,0,2,0e,8,-11,-3,0
*08ADF,31,unspecified
2,0e,8,-7,-15,
7,0,089a4,0,0,8,16,7,0,08cb3,8,0,8,16,
2,8,18,0,2,0e,8,-11,-3,0
```

```
*08CE8,39,unspecified
2,0e,8,-7,-15,
7,0,089a4,0,1,5,14,7,0,08cc3,5,2,5,14,7,0,08c8e,9,0,7,
16,2,8,18,0,2,0e,8,-11,-3,0
*089A4,39,primitive
2,0e,8,-7,-15,2,8,1,14,1,0c0,
2,8,-11,-6,1,0a0,2,8,-12,-7,1,
0e0,2,8,-7,13,1,0dc,2,8,11,-1,
2,0e,8,-11,-3,0
*08BCA,41,primitive
2,0e,8,-7,-15,2,8,1,14,1,0c0,
2,8,-11,-6,1,0a0,2,8,-12,-8,1,
0e0,2,0e5,1,0ec,2,063,1,8,
2,-3,2,06f,2,0e,8,-11,-3,0
*08BE0,81,primitive
2,0e,8,-7,-15,2,8,3,9,1,080,
2,8,-10,-4,1,0c0,2,8,-13,-5,1,
0e0,2,8,-7,9,1,09c,2,8,-1,14,
1,8,-6,-5,2,8,8,5,1,8,6,-5,
2,8,-11,-6,1,8,1,-3,2,8,7,3,
1,8,-1,-3,2,8,-3,15,1,01a,2,
012,1,01e,2,8,10,-14,2,0e,8,
-11,-3,0
*08C8E,44,primitive
2,0e,8,-7,-15,2,8,3,15,1,090,0fc,038,
2,8,-6,11,1,090,2,8,-9,-5,1,
090,2,096,1,0ac,8,-1,-3,01a,01a,2,8,
18,0,2,0e,8,-11,-3,0
*08CB3,61,primitive
2,0e,8,-7,-15,2,042,1,02b,02a,018,2,
0d0,1,012,034,2,069,1,01e,040,2,8,
-8,6,1,02b,2,8,4,5,1,08c,2,8,
-3,8,1,03c,2,8,-5,3,1,0e0,2,8,
-12,5,1,0a0,2,8,6,-14,2,0e,8,
-11,-3,0
*08CC3,34,primitive
2,0e,8,-7,-15,2,0c1,1,06c,0a8,064,0a0,2,8,
-5,9,1,09c,2,8,-7,5,1,0e0,2,8,
4,-11,2,0e,8,-11,-3,0
*08CFB,22,primitive
2,0e,8,-7,-15,2,0d2,1,0cc,0c8,0c4,0c0,2,8,
5,-13,2,0e,8,-11,-3,0
```

In some drafting disciplines, many special symbols can appear in text strings. The AutoCAD standard text fonts can be extended to include special symbols; however, extending standard text fonts has several limitations:

- The limit is 255 shapes per font file.
- The standard character set uses almost half the available shape numbers. Only codes 1-9, 11-31, and 130-255 are available.
- If you want to use multiple text fonts, you must duplicate the symbol definitions in each font.
- To use a special symbol, you must enter ? ? ***nnn***, where *nnn* is the symbol's shape number.

The big font mechanism avoids these problems. You can select one or more seldom-used characters (such as the tilde (~) or the vertical bar(|)) as escape codes, and use the next character to select the appropriate special symbol. For instance, you can use the following big font file to draw Greek letters by entering a vertical bar (|, ASCII code 124) followed by the equivalent Roman letter. Because the first byte of each character is 124, the character codes are biased by 124 x 256, or 31744.

```
*BIGFONT 60,1,124,124
*0,4,Greek
above, below, modes, 0
*31809,n,uca
. . .    uppercase Alpha definition, invoked by "|A"
*31810,n,ucb
. . .    uppercase Beta definition, invoked by "|B"
*31841,n,lca
. . .    lowercase Alpha definition, invoked by "|a"
*31842,n,lcb
. . .    lowercase Beta definition, invoked by "|b"
*31868,n,vbar
. . .    vertical bar definition, invoked by "||"
. . .
```

Using a Big Font

To use a big font for drawing text, you must set up a text style by using the STYLE command and then specify the name of the big font file. The same text style can use a normal ASCII font as well; enter only the two file names, separated by a comma.

Command: **style**

Text style name (or ?): *name*

Font file <*current*>: **txt,greek**

AutoCAD assumes that the first name is the normal font and that the second is the big font.

If you enter only one name, AutoCAD assumes it is the normal font and removes any associated big font.

By using leading or trailing commas when specifying the font file names, you can change one font without affecting the other, as shown in the following table.

Input for changing fonts

Input	Result
normal, big	Both normal and big font specified
normal,	Normal font only (big font unchanged)
,*big*	Big font only (normal font unchanged)
normal	Normal font only (if necessary, big font removed)
ENTER (null response)	No change

When you use the STYLE command to list styles or to revise an existing style, AutoCAD displays the normal font file, a comma, and the big font file. If the style has only a big font file, it is displayed with a leading comma, as in ,greek.

For each character in a text string, AutoCAD searches the big font file first. If the character is not found there, the normal font file is searched.

Chapter 3 -- Shapes, Fonts, and PostScript Support

Shape Definition Files

Unicode Font Descriptions

The standard AutoCAD fonts correspond to the character mapping used by the host operating system. This is because characters are stored directly in the database in the format in which they are obtained from the keyboard. The same character codes are used to generate fonts. This becomes a problem when using accented (8-bit) characters for which many character encoding standards exist.

Due to character mapping limitations, it has been necessary to provide a set of fonts for the various code pages that AutoCAD uses. These fonts, while essentially the same, have some characters located in different places, depending on the code page they are defined for. If the font encoding used does not match that of the text in the drawing, the wrong characters may be drawn.

With Unicode fonts, text strings are converted to Unicode *before* being drawn, so it is no longer necessary to provide additional fonts for other languages or platforms. A single Unicode font, due to its large character set, is capable of supporting all languages and platforms. This feature is transparent to the user because the drawings are, if needed (due to differing code pages), converted to AutoCAD's system code page at load time. The drawings are always saved in AutoCAD's system code page.

Note Unicode does *not* provide adequate support for all Asian languages, so big fonts are still used by some or all Asian versions.

Topics in this section:

File Format

Unicode shape definition files are virtually identical in format and syntax to regular AutoCAD shape definition files. The main difference is in the syntax of the font header as shown in the following code shows.

```
*UNIFONT,6,font-name
above,below,modes,encoding,type,0
```

The *font-name*, *above*, *below*, and *modes* parameters are the same as in regular fonts. The remaining two parameters are defined as follows:

encoding

Font encoding. One of the following integer values.

0 Unicode
1 Packed multi-byte 1
2 Shape file

type

Font embedding information. Specifies whether the font is licensed. Licensed fonts must not be modified or exchanged. (bit-coded values, can be added)

0 Font can be embedded
1 Font cannot be embedded
2 Embedding is read only

The only other difference between Unifont shape definitions and regular shape definitions is the shape numbers. The Unifont shape definitions that are provided with AutoCAD use hexadecimal shape numbers as opposed to decimal values. This is not required but does make it easier to cross reference the shape numbers with the `\U+` control character values.

Shape Definition Files

Advanced Shape Definition Technique

AutoCAD's supplied fonts are limited as far as superscript and subscript facilities are concerned. However, it is relatively easy to modify shape definition files to improve this facility.

Creating superscripts and subscripts requires two steps. First, the "imaginary pen" that is creating the text, vector by vector, on your screen needs to be shifted up or down. Then, the font "scale" needs to be reduced. In addition, the reverse process has to take place to return to the normal font. So the font needs to recognize four new keys: two for superscripts and two for subscripts. To avoid altering the existing font definitions, you can access these with the numeric keypad on your keyboard.

The following example was created based on AutoCAD's ROMANS font file, although a similar method applies to any AutoCAD font. This procedure adds four new shape definitions to a font: super_on, super_off, sub_on, and sub_off, which control the position and size of the characters that follow. For simplicity, this example replaces the left- and right-bracket characters ([and]) and the left and right curly brace characters ({ and }) with the new characters. You may choose to replace other characters or use a shape number in the extended range (ASCII codes 128 through 256). If you use an extended shape number, you need to use the %%*nnn* method (where *nnn* is the ASCII value of the character) for placing the new characters.

1 Edit the *.shp* (in this case, *romans.shp*) file with an ASCII text editor. You may want to create a new file called *romanss.shp* rather than modifying the original file.

2 Search for the shape definitions of the characters you are replacing. You need to comment out those definitions, so the new definitions can take their place. To comment out the shape definitions, insert a semicolon in front of each line of the shape definition. The shape definition may continue for a number of lines.

   The left- and right-bracket characters have ASCII values of 91 and 93 (05B and 05D hex values, if the font is Unicode). The left and right curly brace characters have ASCII values of 123 and 125 (07B and 07D hex).

3 Add together the first and second values on the second line of the definition, and divide the total by 2 as shown in the following example:

```
*UNIFONT,6,Extended Simplex Roman for UNICODE
21,7,2,0
```

   21 + 7 = 28, then 28 / 2 = 14. This number is used later.

4 Add the following lines to the end of the *.shp* file.

```
*228,8,super_on
2,8,(0,14),003,2,1,0
*227,8,super_off
2,004,2,8,(0,-14),1,0
*222,8,sub_on
2,8,(0,-14),003,2,1,0
*221,8,sub_off
2,004,2,8,(0,14),1,0
```

   Notice the 14 and -14 values in the preceding lines. They are *Y* axis offsets for the imaginary pen. The value 14 is half the maximum height of a character in this font, which is the correct approximation for superscripts and subscripts. This value needs to be calculated for each font file, but you can modify it any way you want.

   Now save the file.

5 Use the COMPILE command to compile the *.shp* file.

Once the shape is compiled and an appropriate style is defined, you can access the new pen-up and pen-down commands by entering the [,], {, and } characters. The [character initiates superscript and the] character returns from superscript to normal. The { character initiates subscript and the } character returns from subscript to normal.