POSTSCRIPT™
Software From Adobe

Adobe Type Manager Software API: Macintosh

Adobe Developer Support

Technical Note # 5072

14 April 1993

Adobe Systems Incorporated

Corporate Headquarters
1585 Charleston Road PO Box 7900
Mountain View, CA 94039-7900
(415) 961-4400 Main Number
(415) 961-4111 Developer Support
Fax: (415) 961-3769

Adobe Systems Eastern Region
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120
Fax: (617) 273-2336

Adobe Systems Europe B.V.
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z-O, Netherlands
+31-20-6511 200
Fax: +31-20-6511 300

Adobe Systems Japan
Swiss Bank House 7F
4-1-8 Toranomon, Minato-ku
Tokyo 105, Japan
+81-3-3437-8950
Fax: +81-3-3437-8968

PN LPS5072

Contents

List of Figures

Adobe Type Manager Software API: Macintosh

1 Introduction

Adobe Type Manager™ software for the Macintosh® allows well over 10,000 Type 1 PostScript™ language fonts to be used transparently by QuickDraw™ to image text on displays and printers. Adobe Type Manager (ATM™) software patches the normal QuickDraw text drawing procedures so that it gets first chance at imaging any text drawn with QuickDraw.

Note *The words 'font' and 'fonts' used throughout this document refer to PostScript language font programs.*

QuickDraw currently does not have a direct interface that allows imaging of rotated, non-uniformly scaled, or skewed Type 1 fonts. ATM software does have these capabilities. Applications can use a special application programmer's interface (API) to the ATM software to obtain these results.

This document describes the functions of the API that will allow an application to perform the text manipulations mentioned above as well as functions associated with font substitution. The remaining functions in the API, multiple master font related functions, are described in Technical Note #5074, "Adobe Type Manager Software API for Multiple Master Fonts: Macintosh."

1.1 A Note to Previous ATM Software API Users

ATM software version 3.5 (API procset version 8), also called SuperATM™, is the first ATM software to include font substitution-related functions. ATM software Japanese version (API procset version 7) is the first and only current ATM software to include functions to be used with characters in the Japanese font products. ATM software version 3.0 (API procset version 5) is the first ATM software to include Multiple Master related functions. The interface (*ATMInterface.c* and *ATMInterface.h*) that supports these functions also supports the older procsets and will work with the older ATM software versions. Note however, that this interface has slightly different function names than the previous interface versions.

For example, **ATMShowText( )** is now called **showTextATM( )**. Additionally, a few of the parameter types have changed. For example, **showTextATM( )** takes a (*Byte **) where **ATMShowText( )** took a (*char **). If the Multiple Master functions are to be fully exploited, the application must be updated to match the new interface files. See Appendix C for tips on converting from pre-version 5 API procsets.

1.2 Functionality

The API has a number of function calls that will be discussed here. Using the **initVersionATM( )** call, an application first checks whether the ATM software is running and which versions of the API functions are available. **fontAvailableATM( )** is a quick check of whether the ATM software will be able to image the font and style you request. **showTextATMErr( )** allows an application to pass a string and a matrix to the ATM software. (The *Err* suffix differentiates it from an older show text call that didn't return meaningful error codes.) The string will be imaged with the font set by the QuickDraw call **TextFont( )**, at the size and orientation given by the matrix.

For those familiar with the capabilities of the PostScript operator **makefont**, the matrix passed to **showTextATMErr( )** is similar and is described later in more detail.

Another function, **xyShowTextATMErr( )** extends the functionality of **showTextATMErr( )** by allowing each character in the string to be positioned explicitly. This allows applications to carefully position each character. Next there is **getOutlineATM( )**, which allow an application to retrieve the Type 1 outline description of a character in Type 1 font. Complementing **getOutlineATM( )** is a set of calls that includes routines to allow applications to define and fill an arbitrary path, which can include Bézier curves. **getOutlineATM2( )** has the same functionality as **getOutlineATM( )**, and additionally allows applications to obtain outline data of double-byte encoded characters in Japanese fonts.

In order to support SuperATM functionality by the API, **isSubstFontATM( )** and **getPSNumATM( )** functions have been added. **isSubstFontATM( )** determines if a specified font is a substitute font, while **getPSNumATM( )** creates a faux font and provides its font ID. Technical descriptions about these functions are provided in this document.

Finally, there are several (over 20) functions that allow applications to take full advantage of the multiple master font technology. These multiple master related functions are described in Technical Note #5074, "Adobe Type Manager Software API for Multiple Master Fonts: Macintosh."

2 Getting Started

Begin by calling **initVersionATM( )** to initialize the version of the ATM software API procedures that will be used. At this time, there are five versions of API functions available in ATM software. The original version, *version 3*, is available as of ATM software version 1.2; the second, *version 4*, is available as of ATM software version 2.0; the third, *version 5*, is available as of ATM software version 3.0; the fourth, *version 7*, is available as of ATM software Japanese version 2.0.x; finally the fifth, *version 8*, is available as of ATM software version 3.5.

This document describes all non-multiple master specific functions in all the API versions. Note that below each function description, there is a line that describes which ATM software version(s) contains that function. For example, the line

```
Version 3 and above
```

would follow the **initVersionATM( )** heading indicating that all API versions 3 and above support the function **initVersionATM( )**.

```
version 3, 4, and 5
```

implies that an API function is supported only in API versions 3, 4, and 5.

Note *ATM software preserves the application's registers A1-A5 and D1-D7 across calls to the API.*

initVersionATM short initVersionATM (short apiVersion)

Version **3** and above

initVersionATM( ) checks whether the ATM software is running and initializes the API functions.

apiVersion is the API version number being requested.

initVersionATM( ) returns 1 if the ATM software is running and the desired version of the API routines is available. Otherwise, 0 is returned.

initVersionATM( ) first checks whether the ATM software is installed by making the **OpenDriver( )** call with the driver name *.ATM*. (An application or print driver, which only checks whether the ATM software is running, need only make this **OpenDriver( )** call.) If the driver is available, **initVersionATM** sets up a driver status call to the ATM software driver with *csCode = ATMProcsStatusCode* and *csParam* set to a pointer to a procs data structure.

Before the status call, the procs data structure is also filled in with the version of the API procset being requested. This allows expandability in the future. As of this writing, the available procsets are version 3 (ATM 1.2), version 4 (ATM 2.0), version 5 (ATM 3.0), version 7 (ATM Japanese version 2.0.x), and version 8 (ATM 3.5). A **PBStatus** call is then made with the appropriate data structure passed in. If the procset version requested is available, the **PBStatus** call returns with no error and the procset data structure is filled in appropriately. If the procset version requested is not available, **PBStatus** returns an error.

If the application requires the Multiple Master related functions, *apiVersion* must be 5 or greater. If version 5 is not available but the application wants to render text through the API, the application can then make the call **initVersionATM(***apiVersion***)** where *apiVersion* is a lower number. The application must be coded to properly handle the lower version (that is, don't call API functions that are not in the currently running version).

If the application's functionality is limited because of the version of ATM software or because there is no ATM software running on the system, the application should notify the user of the problem.

fontAvailableATM

short fontAvailableATM (short family, short style)

Version **3** and above

fontAvailableATM( ) is a quick check of whether the ATM software might be able to image in a given typeface. **fontAvailableATM( )** is passed a font family number and a QuickDraw style code.

family is the Macintosh font family number (the number passed to **TextFont( )**).

style is the QuickDraw style number (like the number passed to **TextFace( )**).

fontAvailableATM( ) returns 1 if there is a PostScript language outline font file available that can be used to image the requested *family* and *style*. Otherwise, 0 is returned.

This call takes the font family number and style passed in and looks at the style mapping table of the FOND resource to determine which PostScript language fonts it can use to image that request. For example, a request for Palatino* bold can be rendered by either the Palatino-Bold outline font or the Palatino-Roman font made bold algorithmically.

If a file exists that corresponds to one of the PostScript language fonts that the ATM software can use to image the requested *family* and *style* (via **showTextATMErr( )** or **xyShowTextATM( )**), **fontAvailableATM( )** returns 1, otherwise 0 is returned.

Be aware that this call does not actually parse the file to determine if it is a valid Type 1 font that the ATM software can use; rather it checks only for the existence of the file.

Note *Unlike* ***fontAvailableATM( )****,* ***getOutlineATM( )*** *requires that the outline of the actual requested style be present and does not take into account that ATM software can algorithmically stylize the regular font outline if available.*

2.1 Checking Whether ATM Software Can Image a Font

An application or print driver can reliably test whether the ATM software can image a font by imaging a character in the font with **showTextATMErr( )** and looking at the value returned. If **showTextATMErr( )** returns 0, then the font is available. This should be done without imaging the character into a visible bitmap.

In addition, take care to ensure that the character code chosen corresponds to a true character in the font. If this is not the case, **showTextATMErr( )** will return a non-zero value and the application will act as though the font is *not* available. See section 3, “Drawing Text With the ATM Software API,” for a description of **showTextATMErr( )**.

Note *The above check does not allow an application to determine whether an imaged character is from a stylized font outline or from an algorithmically stylized regular font outline. Currently, there is no way to get this information through the API. This might change in the future.*

3 Drawing Text With the ATM Software API

There are three methods of showing text with the ATM software API. Two methods are described in this section. The first, **showTextATMErr( )** is the basic function that allows rotation, scaling, and skewing of text. Secondly, in addition to these capabilities, **xyShowTextATMErr( )** allows kerning and special effects through *x* and *y* displacements for each character in the string.

Technical Note #5074, "Adobe Type Manager Software API for Multiple Master Fonts: Macintosh," discusses **showTextDesignATM( )**, which is a method of efficiently previewing multiple master fonts.

showTextATMErr

```
short showTextATMErr (
        Byte*           text,
        short           length,
        ATMFixedMatrix *matrix,
        short           *errorCode )
```

Version **4** and above

showTextATM

```
short showTextATM (
        Byte        *text,
        short       length,
        ATMFixedMatrix *matrix )
```

Version **3**

showTextATMErr( ) allows an application to image type in any orientation, skew, and scale including non-uniform scaling.

text* is a pointer to an array of characters to be imaged. (Pascal programmers: skip the first element of your string, the length.) The characters in the array are shown relative to the current QuickDraw pen location with the orientation and scale as specified by *matrix*. The font used to draw the string is that selected by QuickDraw's **TextFont( ) and **TextFace( )** functions.

length is a count of the number of characters in the string to be imaged.

**matrix* points to a transformation matrix that can be used to rotate or skew the text. This matrix is described in section 3.1, "The Matrix."

errorCode is set to ATM_NOERR if the call was successful. Otherwise *errorCode* is set to the appropriate ATM software error code. See the error code listing in Appendix A.

showTextATMErr( ) returns the number of characters in the string that were *not* imaged.

showTextATM( ) is similar to **showTextATMErr( )**. The only difference is that there is no parameter corresponding to *errorCode*.

3.1 The Matrix

The matrix passed in is a structure of six *Fixed* point numbers. Fixed point numbers represent floating point numbers in the 32K range and can be converted to a floating point number by dividing by 65536. This matrix is similar (but not identical) to the matrix passed to the PostScript operator **makefont**. Appendix B discusses the differences between this transformation matrix and that normally passed to the **makefont** operator.

The matrix structure looks like this

```
typedef struct
{
   Fixed a,b,c,d,tx,ty;
} ATMFixedMatrix;
```

This matrix maps the coordinate system you want onto the QuickDraw coordinate system. The font is imaged by default in a coordinate system where the origin of coordinates is the current QuickDraw pen position, one unit is 1 screen pixel (1/72 of an inch), and the axes are a standard Cartesian coordinate system (+x to the right, +y goes up the screen).

The matrix values to be passed should be as follows:

Equation 1:

$$a = Mx$$
$$b = -Mx \tan(\alpha)$$
$$c = My \tan(\beta)$$
$$d = -My$$

The elements *a* and *d* are the horizontal and vertical scaling factors. *Mx* is the font size in the x direction. If *Mx* is 12 then the *x* dimension of the font will be that of a 12 point font. *My* is the font size in the *y* direction. The *y* dimension of the font will be that of an *My* point font. The elements *b* and *c* determine the orientation of the axes relative to a normal Cartesian coordinate system. Element *b* includes the skew of the axis in the *x* direction. Figure 1 illustrates the geometry. If α is the upward skew of the transformed *x* axis relative to the normal x axis, then $b = -Mx \tan(\alpha)$. Similarly, if β is the clockwise vertical skew of the transformed y axis relative to the normal *y* axis, then $c = My \tan(\beta)$.

Note that these values have been adjusted to map into the QuickDraw coordinate system.

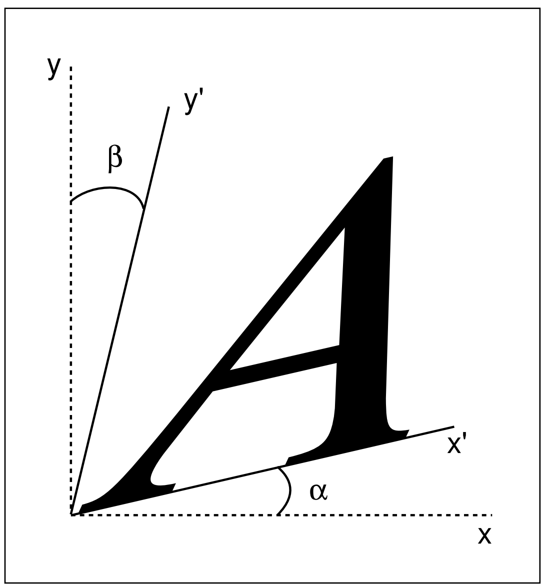
A slightly different formula is easier for uniform rotations. If a point size of *M* points is desired with a counterclockwise rotation of θ degrees, then

Equation 2:

$$a = M\cos\theta$$
$$b = -M\sin\theta$$
$$c = -M\sin\theta$$
$$d = -M\cos\theta$$

Finally, the *tx* and *ty* values of the matrix are translation components in the *x* and y directions. These units are in the QuickDraw coordinate system, where +x goes across to the right and +y goes down the screen. This positions the origin of the first character imaged at *tx* units over to the right and *ty* units down from the QuickDraw pen position.

Figure 1 *Skew of the coordinate system*



The skew of the coordinate system is specified by the parameters α and β. The directions of skew indicated in the drawing would result in positive values of both α and β.

Examples

The following pages contain some simple code examples showing how the matrix should look when passed into **showTextATMErr( )**.

Figure 2 *Upright text*

Example 1:

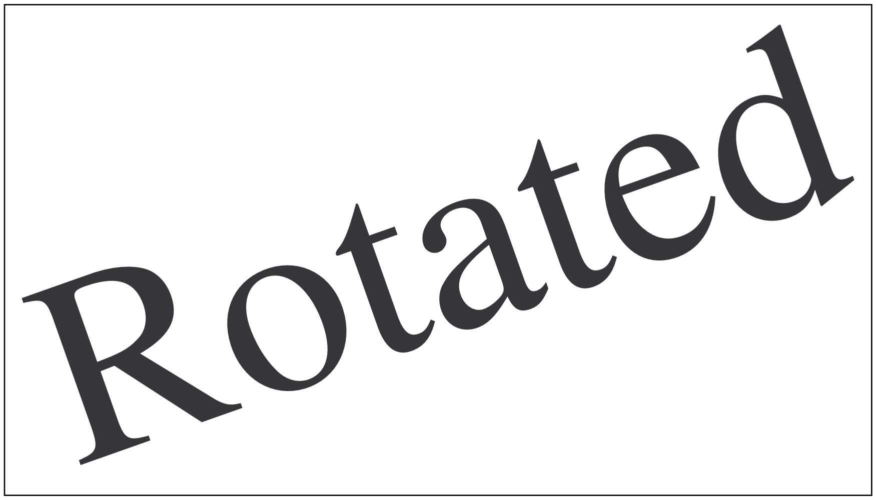
```
    ATMFixedMatrix m;
    Byte *c;
    short unrendered;
    short errCode;
    short version = 5;

  /* be sure and set a Type 1 PostScript language font
    with TextFont( )
    passing in 20 gives Times-Roman in most cases but depends
    on what bitmapped fonts the user has installed
  */
    TextFont(20);

  /*initialize the ATM software. If 0 is returned, we must punt.
  */
    if (!initVersionATM(version));
              return 0;

  /* Numbers are converted to fixed by multiplying by 65536.
    ATM truncates the tx and ty components of the matrix. To
    compensate for this we add 0.5,take the floor then multiply
    by 65536. See the code below
  */

    m.a = (ATMFixed) 226.7 * 65536.0;
    m.b = (ATMFixed) 0.;
    m.c = (ATMFixed) 0.;
    m.d = (ATMFixed) -226.7  * 65536.0;
    m.tx = (ATMFixed) floor (100.3 + 0.5) * 65536.0;
    m.ty = (ATMFixed) floor (200.4 + 0.5) * 65536.0;

    MoveTo(0, 0);
    c = "A";
    unrendered = showTextATMErr(c, 1, &m, &errCode);
```

Figure 3 *Reversed text*

Example 2:

```
/* reversed in x. */
#define FIX(q) (ATMFixed)  floor (q + 0.5) * 65536.0
ATMFixedMatrix m;
Byte  *c;
short unrendered;
short errCode;
short version = 5;
/*
   be sure and set a Type 1 PostScript language font with
   TextFont() (and TextFace() if stylized)
*/
   TextFont(20);

/* initialize the ATM software. If 0 is returned, we must punt.
*/
if (!initVersionATM (version))
     return 0;

/*
   minus sign in 1st component of matrix flips coordinates so text
   is scaled in -x; this is equivalent to mirrored along y axis
*/

   m.a = (ATMFixed) -79.7 * 65536.0;
   m.b = (ATMFixed) 0.;
   m.c = (ATMFixed) 0.;
   m.d = (ATMFixed) -204.6 * 65536.0;
      m.tx = FIX(400.3);
   m.ty = FIX(200.4);

   MoveTo(0, 0);
   c = "Reversed";
   unrendered = showTextATMErr(c, 8, &m, &errCode);
```

Figure 4 *Rotated text*



Example 3:

```
    #define    DEGTORAD    (2*3.14159/360)
      /*   2pi/360 */

    #define FIX(q) (ATMFixed)  floor (q + 0.5) * 65536.0

    ATMFixedMatrix m;
    Byte  *c;
    short unrendered;
    short errCode;
    short version = 5;
  /*
    be sure and set a Type 1
    PostScript language font with TextFont()
  */
    TextFont(20);

  /*
    initialize the ATM software. If 0 is returned, we must punt.
  */
    if (!initVersionATM (version))
      return 0;
    m.a= (ATMFixed)  44.6*cos(25*DEGTORAD) * 65536.0;
    m.b= (ATMFixed) -44.6*sin(25*DEGTORAD) * 65536.0;
    m.c= (ATMFixed) -44.6*sin(25*DEGTORAD) * 65536.0;
    m.d= (ATMFixed) -44.6*cos(25*DEGTORAD) * 65536.0;
    m.tx = FIX (60.3);
    m.ty = FIX (110.4);

    MoveTo(0, 0);
    c = "Rotated";
    unrendered = showTextATMErr(c, 7, &m, &errCode);
```

Figure 5 *Skewed text*

Example 4:

```
/* 44.6 point text in x, skewed 15 degrees up, 83.2 point text in
   y, skewed 35 degrees clockwise. */
#define DEGTORAD  (2*3.14159/360)
   /*   2pi/360 */
#define FIX(q) (ATMFixed)  floor (q + 0.5) * 65536.0
ATMFixedMatrix m;
Byte  *c;
short unrendered;
short errCode;
short version = 5;
/*
   be sure and set a Type 1
   PostScript language font with TextFont()
*/
TextFont(20);

/*
   initialize the ATM software. If 0 is returned, we must punt.
   This initialization should only be done once per program
*/
if (!initVersionATM (version))
     return 0;

m.a = (ATMFixed) 44.6 * 65536.0;
m.b = (ATMFixed) -44.6* tan(15*DEGTORAD) * 65536.0;
m.c = (ATMFixed) 83.2 * tan(35*DEGTORAD) * 65536.0;
m.d = (ATMFixed) -83.2 * 65536.0;
m.tx = FIX (50.3);
m.ty = FIX (120.4);
MoveTo(0, 0);
c = "Skewed";
unrendered = showTextATMErr(c, 6, &m, &errCode);
```

3.2 showTextATMErr/showTextATM (Continued)

showTextATMErr( ) will image the characters in the string one by one. The translation components *tx* and *ty* are adjusted after each character is imaged so they indicate (in QuickDraw screen coordinates) where the origin of the next character will be relative to the QuickDraw drawing pen. The QuickDraw drawing pen is not moved by **showTextATMErr( )**.

showTextATMErr( ) returns the number of characters in the string that were not imaged. If **showTextATMErr( )** returns 0, then all the characters in the string were imaged. If **showTextATMErr( )** cannot image a character in the string passed in, then **showTextATMErr( )** will properly image all characters before that character and then will return.

The value of the *tx*,*ty* components of the matrix are updated to the point where the unshown character in the string would have been shown, had there been a corresponding character glyph in the font. This is done so an application program can take its own steps to image the character, which the ATM software could not.The usual reason that the ATM software cannot image a given character is that the character not imaged does not correspond to a character in the current font. (QuickDraw would normally image the box character.) For most fonts, the $CA, $7F, and $00-$1F character codes do not have characters associated with them.

3.3 Potential Problems

The following are other reasons that might cause the text showing routines to return prematurely:

- The font requested is not a valid Type 1 font. For example, some fonts created with early font creator applications are Type 3 fonts. The current versions of the ATM software only know how to render Type 1 fonts. Version 4 and above API routines will set the *errorCode* to *ATM_NO_VALID_FONT*.

- The current ATM software can only image characters that will fit in the font cache. If a given character cannot fit in the font cache, then **showTextATMErr( )** will return prematurely. Version 4 and above API routines will set the *errorCode* to *ATM_MEMORY*.

- **showTextATMErr( )** expects to be passed a valid matrix. The ATM software supports a scaling factor of 8000 or 255 times the size of the largest bitmap point size installed in the system, whichever is smaller. If the matrix is not valid, then **showTextATMErr( )** will return without imaging any characters. Version 4 and above API routines will set the *errorCode* to *ATM_BAD_MATRIX*.

- The ATM software periodically checks the consistency of its data structures. If another process on the Macintosh writes into these data structures, then the ATM software will turn itself off. This will cause calls to **showTextATMErr( )** to return without any characters imaging. Version 4 and above API routines will set the *errorCode* to *ATM_CANT_HAPPEN*.

xyShowTextATMErr

```
short xyShowTextATMErr (
        Byte*               text,
        short               length,
        ATMFixedMatrix      *matrix,
        ATMFixed            *displacements,
        short               *errorCode )
```

Version **4** and above

xyShowTextATM

```
short xyShowTextATM (
        Byte*           text,
        short           length,
        ATMFixedMatrix  *matrix,
        ATMFixed        *displacements )
```

Version **3**

xyShowTextATMErr( ), like **showTextATMErr( )**, provides an application with the ability to image type in any orientation, skew, and scale, including non-uniform scaling. Additionally, **xyShowTextATMErr( )** takes an array of numbers, which are displacements in the *x* and *y* directions, for each character in the string. The font used to draw the string is that selected by QuickDraw's **TextFont( )** and **TextFace( )** functions.

text* is a pointer to an array of characters to be imaged (Pascal programmers: skip the first element of your string, the length). The characters in the array will be shown relative to the current QuickDraw pen location with the orientation and scale as specified by *matrix*. The font used to draw the string is that selected by QuickDraw's **TextFont( ) and **TextFace( )** functions.

length is a count of the number of characters in the string to be imaged.

**matrix* points to a transformation matrix that can be used to rotate or skew the text. This matrix is described in section 3.1, "The Matrix."

**displacements* is a pointer to an array of fixed point numbers, which are displacements in the *x* and *y* directions, for each character in the string. See below for further information on the displacement array.

errorCode is set to *ATM_NOERR* if the call was successful. Otherwise *errorCode* is set to the appropriate ATM software error code. See the error code listing in Appendix A.

xyShowTextATMErr( ) returns the number of characters in the string that were *not* imaged.

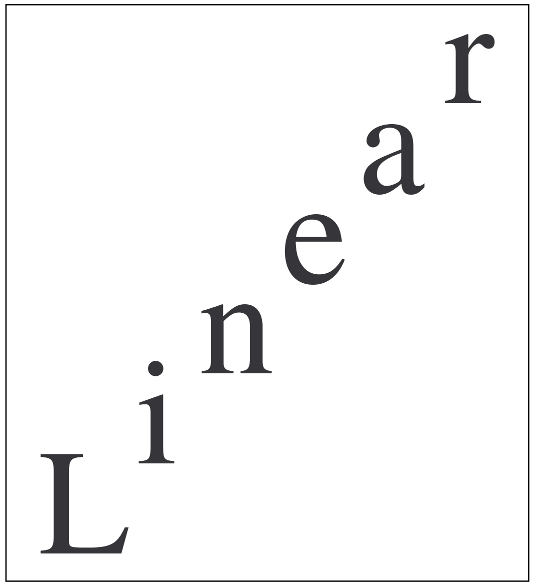
3.4 Displacement Array

The number of elements in the array is twice the number of characters being imaged. This ‘displacements’ array is a series of *x* and *y* widths for the characters being imaged in the character array. It is possible to have different widths used for a given character code that appears more than once in the character array. This allows for both track kerning and pairwise kerning, as well as special effects.

The 0th element of the array is the *x* width to be used for the first character. The next element in the array is the *y* width for the first character, and so on. For the *i*th element of the character array, the width used for that character is **wx** = **displacements[2i]**, **wy** = **displacements[2i+1]**. The numbers in the array are the widths as if the font were being imaged as a 1 point font. They will be appropriately scaled (and rotated, and so on) by the transformation matrix passed in to the **xyShowTextATMErr( )** call. For example, this means that for most Roman typefaces, the *y* value will be 0, even when imaging rotated text.

xyShowTextATM( ) is similar to **xyShowTextATMErr( )**. The only difference is that there is no parameter corresponding to *errorCode*.

Figure 6 *Text using the displacement array*



Example 5:

```
/* Chars distributed on a line of slope 1 with constant widths */
#define FIX(q) (ATMFixed)  floor (q + 0.5) * 65536.0
ATMFixedMatrix m;
Byte  *c;
Fixed d[12];
short loop,unrendered;
short errCode;
short version = 5;
/*   be sure to set a Type 1
   PostScript language font with TextFont( )
*/
TextFont(20);
/*   initialize the ATM software. If 0 is returned, we must punt.
*/
if (!initVersionATM (version))
     return 0;
m.a = (ATMFixed) 25* 65536.0;
m.b = (ATMFixed) 0.;
m.c = (ATMFixed) 0.;
m.d = (ATMFixed) -25 * 65536.0;
m.tx = FIX (10.);
m.ty = FIX (180.);
/*   x and y widths are both 1 for a 1 point font imaged
     ...with slope 1 */
for (loop = 0; loop<12; loop++)
     d[loop] = (ATMFixed) 1 * 65536.0;
MoveTo(0, 0);
c = "Linear";
unrendered = xyShowTextATMErr(c, 6, &m, d, &errCode);
```

4 Obtaining a Character's Outline Description

The notation *FIX()* is used throughout the remainder of this document. It is defined as

```
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0
```

getOutlineATM

```
short getOutlineATM (
    short     ch,
    ATMFixedMatrix*     matrix,
    Ptr                 clientHook,
    short     (*MoveTo)     (/* Ptr clientHook, ATMFixedPoint *pt */),
    short     (*LineTo)     (/* Ptr  clientHook, ATMFixedPoint *pt */),
    short     (*CurveTo) (/* Ptr clientHook, ATMFixedPoint *pt1,*pt2,*pt3) */,
    short     (*ClosePath)  (/* Ptr clientHook */) )
```

Version **4** and above

getOutlineATM( ) provides applications with the capability to get the PostScript language Type 1 font definition of a character outline. The outline is returned as a series of points in the QuickDraw coordinate system that can be used to reconstruct the character using QuickDraw or using the *moveto*, *lineto*, *curveto*, and *closepath* operations (described in section 5).

ch is the character whose outline is to be retrieved.

matrix allows for the transformation (rotation, skewing, and so on) of the character outline returned. This matrix is the same as the matrix used by **showTextATMErr( )** and is described in section 3.1, "The Matrix." Note however, that when using **showTextATMErr( )**, the current QuickDraw pen location is relevant.

This is *not* the case for **getOutlineATM( )**. With **getOutlineATM( )**, the outline is returned with the character origin at the QuickDraw coordinate location specified by the *m.tx* and *m.ty* values of the matrix *m* regardless of the current QuickDraw pen location. This is similar to using **showTextATMErr( )** with the QuickDraw pen location at 0,0.

For example, if the matrix has *m.a = FIX (50.0), m.b = m.c = (ATMFixed) 0, m.d = -m.a, m.tx= FIX (100.0) and m.ty = FIX (100.0)*, the points returned by **getOutlineATM( )** will be the QuickDraw coordinates of regular (not skewed or altered) 50 point character with its origin at the point (100,100). If it is known that the outline need only be displayed at point (100, 100), then *m.tx = m.ty = FIX (100.0)* is fine. However, if the outline can be displayed arbitrarily, it may be best to call **getOutlineATM( )** with *m.tx = m.ty = (ATMFixed) 0* and perform transformations as necessary.

clientHook is a 32 bit value that **getOutlineATM( )** will pass to all the application supplied call-back routines discussed below. **getOutlineATM( )** does not care what the application uses it for. *clientHook* is commonly used as a pointer to a list of data passed to the call-back routines by **getOutlineATM( )**.

MoveTo, *LineTo*, *CurveTo*, and *ClosePath* are call-back routines that correspond to the PostScript operators (**moveto**, **lineto**, **curveto**, and **closepath**) that might be processed by **getOutlineATM( )**. The call-back routines should return 1 if they will to continue being called for the remainder of the current character; they should return 0 if they want to terminate **getOutlineATM( )**.

getOutlineATM( ) returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned. See the error code listing in Appendix A.

getOutlineATM2

```
short getOutlineATM2 (
        char  *ch,
        ATMFixedMatrix*       matrix,
        Ptr                   clientHook,
        short    (*MoveTo)    (/* Ptr clientHook, ATMFixedPoint *pt */),
        short    (*LineTo)    (/* Ptr clientHook, ATMFixedPoint *pt */),
        short    (*CurveTo) (/* Ptr clientHook, ATMFixedPoint *pt1,*pt2,*pt3*/),
        short    (*ClosePath) (/* Ptr clientHook */) )
```

Version **7**, ATM software Japanese version 2.0.x (Japanese version only)

getOutlineATM2( ) provides applications with the capability to get the PostScript language definition of a character outline, for a character with a double-byte character code. **getOutlineATM2( )** is a variant of **getOutlineATM( )** with the same functionality and error codes. All the parameters are the same, except for the first. The first parameter is a pointer to a string, which contains a double-byte character code. **getOutlineATM2( )** returns outline data for the first character code only in the string. The character code is interpreted the same way **DrawText( )** would interpret it, and takes the properties of the currently selected font into account. That is, if the current font is Japanese, the string is interpreted as *Shift-JIS* character codes. If the current font is Roman, the string is interpreted as character codes of the extended ASCII character set used by Macintosh.

The call-back routines should return 1 if they wish to continue being called for the remainder of the current character; they should return 0 if they wish to terminate **getOutlineATM2( )**.

getOutlineATM2( ) returns *ATM_NOERR* for successful operation, after it has enumerated the character path by calling the callbacks. Otherwise, one of the ATM error codes is returned. There are error codes that have been assigned specifically for the Japanese version of ATM software. The error code *ATM_COPY_PROT* is returned when **getOutlineATM2( )** is invoked to obtain outline data for character in Japanese fonts that are copy-protected. When the character outline data of the Japanese fonts are protected, *ATM_PROT_OUTLINE* is returned. Refer to Appendix A for additional ATM software error codes.

Note *Certain Japanese font products, including several from Adobe, are copy-protected or have protected outline data.* ***getOutlineATM2( )*** *will not return outline data for characters in a protected font; instead, it will return an error code, as described above. Currently, at least two Japanese font products, HeiseiMin-W3 and HeiseiKakuGo-W5, are not protected. In addition, there may be PostScript language Japanese typefaces from other vendors which are compatible with ATM software Japanese version and not protected.*

4.1 Details

A PostScript language Type 1 font character can be represented by one of the four basic elements of a path, which are the PostScript operators **moveto**, **lineto**, **curveto** and **closepath**. Each time **getOutlineATM( )** processes one of these, **getOutlineATM( )** executes the corresponding application supplied call-back routine handing the routine *clientHook* and the data that corresponds to the basic element being processed.

As shown in the heading of this section, all of the call-back routines take arguments that include *clientHook* and the appropriate number of pointer(s) to *ATMFixedPoint* points. The **moveto** and **lineto** call-backs each get one *ATMFixedPoint* parameter, the **curveto** call-back gets three *ATMFixedPoint* arguments, and **closepath** requires no *ATMFixedPoint* arguments. The call-back routines passed to **getOutlineATM( )** correspond to the procedure arguments passed to the PostScript operator **pathforall**.

Note that **getOutlineATM( )** returns paths in the same order in which the font designer created the character. This means that an application can not depend on the outer path being returned before the inner path. Additionally, be aware that the path returned by **getOutlineATM( )** does not incorporate any “hints” that might be in the font outline description.

In the current version of ATM software, to get a character outline, the outline of the requested style must be available to **getOutlineATM( )**. This differs from **fontAvailableATM( )**, **showTextATMErr( )**, and **xyShowTextATM( )** where the ATM software will algorithmically stylize the regular outline if it is available. See section 2 for details. If **getOutlineATM( )** requests a font style (for example, **TextFace(***bold***)**) whose outline font is not available, the ATM software error code *ATM_NO_VALID_FONT* is returned by **getOutlineATM( )**.

A typical use of the data returned by **getOutlineATM( )** might be to have the application alter the data before sending the new data to the ATM software fill routines (see section 5, “ATM software Path Rendering and Filling Routines”) or to use the data to create some type of wire frame (for example, a 3-D font) and image the frame using QuickDraw.

Note *As mentioned in the section “Getting Started,” ATM software preserves the applications registers A1-A5 and D1-D7 across calls to the API. Note however, that within the call* ***getOutlineATM( )****, only register A5 is preserved across calls from* ***getOutlineATM( )*** *to the call-back routines. Refer to the ATM software Programmer’s Disk for a detailed example using* ***getOutlineATM( )****.*

Example 6:

```
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0
#define LINE_TO  'lt'
#define FAILURE   0
#define SUCCESS   1
• • •
typedef struct           /* defined in ATMInterface.h */
{
  ATMFixed x, y;
  ATMFixedPoint;
• • •
typedef struct LineToRecord
{           /* represents lineto data */
    short   op_code;        /* constant "LINE_TO "    */
    ATMFixedPoint  pt;         /* store returned point here  */
} LineToRecord;
• • •
static short LineTo ( path_hndl, pt )
Handle  path_hndl;
ATMFixedPoint   *pt;
{
  LineToRecord   ltR;
  OSErr err;
  ltR.op_code = LINE_TO;
  ltR.pt = *pt;
  err = PtrAndHand((Ptr) &ltR, path_hndl, sizeof(ltR));
  return ( err ? FAILURE : SUCCESS);
}
• • •
/* create routines similar to LineTo for MoveTo, CurveTo and
  ClosePath
*/
• • •
Handle path_hndl = NewHandle(0);
ATMFixedMatrix m;
m.a = (ATMFixed) 226.7 * 65536.0; /* 226.7 pt char */
m.b = (ATMFixed) 0.;  /* not skewed or altered  */
m.c = (ATMFixed) 0.;
m.d = -m.a;
m.tx = FIX (100.3);   /* relative to device location */
                         /* ...(100.3, 200.4) */
```

```
m.ty = FIX (200.4);
/* get the outline for the character ’A’ */
err = getOutlineATM(’A’, &m, (int32) path_hndl, MoveTo,
                       LineTo, CurveTo, ClosePath);
• • •
/* End up with a list of MoveTo, LineTo, CurveTo and ClosePath
   records.
*/
```

5 ATM Path Rendering and Filling Routines

Several routines are available that allow an application to use ATM to create and fill paths with the foreground color implicit in the current QuickDraw *grafPort*.

The ATM software fill routines **startFillATM( )** and **endFillATM( )** must surround all calls to the path construction routines **fillMoveToATM( )**, **fillLineToATM( )**, **fillCurveToATM( )**, **fillClosePathATM( )**. All the fill routines return *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned. For more information on ATM software error codes, see Appendix A.

Note that these fill routines closely parallel the PostScript operators **moveto**, **lineto**, **curveto**, **closepath** and **fill**, which are used to create and fill shapes in the PostScript language. See the *PostScript Language Reference Manual, Second Edition* for more details.

5.1 Concepts Used in This Section

point – all references to a *point* in this discussion refer to a point in the QuickDraw coordinate system.

current path – the path being built since the last call to **startFillATM( )**.

subpath – a sequence of connected segments; the *current path* is made up of one or more disconnected subpaths. The operator **fillClosePathATM( )** explicitly connects the end of a subpath back to its starting point. Such a subpath is said to be *closed*.

current point – the trailing endpoint of the most recently constructed path segment. The path construction operators **fillLineToATM( )**, **fillCurveToATM( )**, and **fillClosePathATM( )** add a segment to the *current path* starting at the *current point*. If no current point exists, a call to one of these operators will cause an error.

startFillATM short startFillATM (void)

Version **4** and above

startFillATM( ) places ATM software in a state ready to accept the rendering commands **fillMoveToATM( )**, **fillLineToATM( )**, **fillCurveToATM( )** and **fillClosePathATM( )**. When this state is entered, random alterations to the QuickDraw graphics state are not recommended and the results are undefined. Note however that the QuickDraw *ForeColor* may be changed before **endFillATM( )** is executed. Additionally, when ATM software is in the "fill" state, **getOutlineATM** should not be invoked. **startFillATM( )** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

endFillATM short endFillATM (void)

Version **4** and above

endFillATM( ) paints the area enclosed by the current path (the path drawn since the last **startFillATM( )**) with the foreground color implicit in the current QuickDraw grafPort. The inside of the current path is determined by the PostScript language even-odd rule. See the *PostScript Language Reference Manual, Second Edition* for details of the even-odd rule. **endFillATM( )** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

fillMoveToATM short fillMoveToATM (ATMFixedPoint *pt)

Version **4** and above

fillMoveToATM( ) establishes a new current point without adding a segment to the current path. **fillMoveToATM( )** takes a pointer to an *ATMFixedPoint*, which represents an *x,y* location in the current QuickDraw grafPort. **fillMoveToATM( )** starts a new subpath of the current path and sets the current point to *x,y*. **fillMoveToATM( )** must be called at least once to establish a starting point before any calls to **fillLineToATM( )** or **fillCurveToATM( )** are executed. **fillMoveToATM( )** is similar to the PostScript language **moveto** operator. **fillMoveToATM( )** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

fillLineToATM short fillLineToATM (ATMFixedPoint *pt)

Version **4** and above

fillLineToATM( ) takes a pointer to an *ATMFixedPoint*, which represents an *x,y* location in the current QuickDraw grafPort and appends a straight line segment from the current point to the point (x,y). The point (x,y) becomes the new current point. **fillLineToATM( )** is similar to the PostScript language **lineto** operator. **fillLineToATM** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

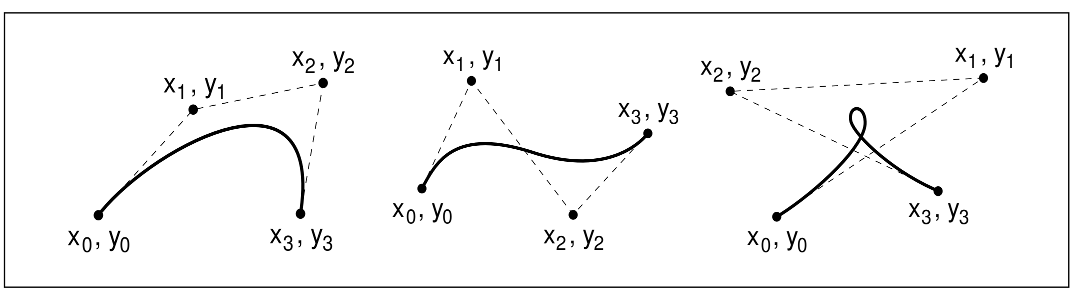
fillCurveToATM short fillCurveToATM(

```
        ATMFixedPoint      *pt1,
        ATMFixedPoint      *pt2,
        ATMFixedPoint      *pt3 )
```

Version **4** and above

fillCurveToATM( ) adds a section of a Bézier cubic curve to the current path. It takes three pointers to *ATMFixedPoint* points, which represent the locations (x_1,y_1), (x_2,y_2), (x_3,y_3) in the current QuickDraw grafPort, and adds a Bézier cubic section to the current path between the current point, referred to here as (x_0, y_0) and the point (x_3, y_3), using (x_1, y_1) and (x_2, y_2) as the Bézier cubic control points. The point (x_3,y_3) becomes the new current point. **fillCurveToATM( )** is similar to the PostScript language **curveto** operator. **fillCurveToATM( )** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

Figure 7 *Curves showing Bézier control points and end points*



The mathematical formulation of a Bézier cubic curve is derived from a pair of parametric cubic equations.

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_0$$
$$y(t) = a_y t^3 + b_y t_2 + c_{yt} + y_0$$

The points returned by **fillCurveToATM( )** represent the path traced by x(t) and y(t) as t ranges from 0 to 1. The Bézier control points corresponding to this curve are

$$x_1 = x_0 + c_x/3 \qquad y_1 = y_0 + c_y/3$$
$$x_2 = x_1 + (c_x + b_x)/3 \qquad y_2 = y_1 + (c_y + b_y)/3$$
$$x_3 = x_0 + c_x + b_x + a_x \qquad y_3 = y_0 + c_y + b_y + a_y$$

fillClosePathATM Short fillClosePathATM (void)

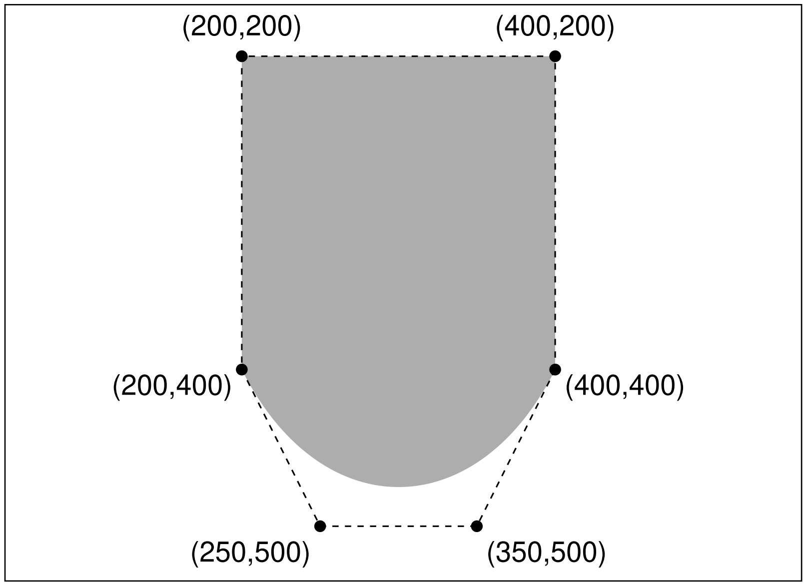
Version **4** and above

fillClosePathATM( ) closes the current subpath by appending a straight line segment connecting the current point to the subpath's starting point (generally, the point most recently specified by **fillMoveToATM( )**). **fillClosePathATM( )** returns *ATM_NOERR* if the call was successful, otherwise the appropriate ATM software error code is returned.

Example 7:

```
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0
short err;
ATMFixedPoint pt1,pt2,pt3;
ForeColor( redColor );
err = startFillATM();
pt1.x = (ATMFixed) 200.0* 65536.0; /* could do this for
               floats or Long2Fixtoolbox function ...*/
pt1.y = Long2Fix(200);                    /* ... for longs */
err = fillMoveToATM( &pt1 );
pt1.x = Long2Fix( 200);
pt1.y = Long2Fix( 400);
err = fillLineToATM( &pt1 );
pt1.x = Long2Fix(250);
pt1.y = Long2Fix(500);
pt2.x = Long2Fix(350);
pt2.y = Long2Fix(500);
pt3.x = Long2Fix(400);
pt3.y = Long2Fix(400);
err = fillCurveToATM( &pt1, &pt2, &pt3 );
pt1.x = Long2Fix(400);
pt1.y = Long2Fix(200);
err = fillLineToATM( &pt1 );
err = fillClosePathATM();
err = endFillATM();
• • •
/* See Figure 8 */
```

Figure 8 *Path Rendering and Filling*



The gray filled area was drawn by the above code. The dashed line represents the path and the points represent the line end points and the Bézier control points in the QuickDraw coordinate system.

6 SuperATM APIs

The Macintosh® Toolbox call **GetFNum( )**, when given a font name, returns a font number that can be used with QuickDraw® functions such as **TextFont( )**.

Like ATM version 3.0 which supports multiple master font technology, SuperATM checks what font is being requested by patching **GetFNum( )** In addition to fully supporting multiple master fonts, SuperATM also provides "on-the-fly" (automatic) font substitution for about 1500 fonts. If a requested font is not available on the system, and its metrics are included in Super-ATM's font database, SuperATM will create a substitute font (referred to in this document as a faux font) with equivalent metrics and weight.

Applications that don't store menu names to call **GetFNum( )**with can use the following functions to create faux fonts via a PostScript language name.

isSubstFontATM boolean

isSubstFontATM (	StringPtr	fontName,
	short	fondID,
	short	style,
	FontSpecs	*** fontSpecs,
	Handle	*fauxName)

Version **8**

isSubstFontATM( ) determines if the font identified by *fontName* or *fondID* is a substitute font. If it is, then additional information about the font is provided.

fontName Specifies the Macintosh font name that is being queried about. If *fontName* is *NULL*, then *fondID* is used.

fondID Specifies the *FOND ID* of the font that is being queried about. This is used only if *fontName* is *NULL*.

style Specifies the Macintosh style of the font being queried about. style's bit positions are described below.

fontSpecs If **isSubstFontATM( )** == *TRUE* and *FontSpecs* != *NULL*, then *fontSpecs* returns a handle to the *FontSpecs* record for the file. The calling application is responsible for disposing of this handle when finished.

See **getFontSpecsATM( )** in Technical Note #5074 for information on the *FontSpecs st*ructure.

fauxName If **isSubstFontATM( )** = *TRUE* and *fauxName* != *NULL*, then *fauxName* returns a handle containing the PostScript language font name of the faux font. The calling application is responsible for disposing of this handle when finished.

isSubstFontATM( ) returns *TRUE* if the font being queried for is a faux font. Otherwise *FALSE* is returned.

Style bit positions have the following values:

Constant	Value	Position
–	0	– regular style, no constant
bold	1	0
italic	2	1
underline	4	2
outline	8	3

shadow	16	4
condense	32	5
extended	64	6

getPSNumATM ATMErr getPSNumATM (StringPtr psName,
short *retFondID,
Boolean doCreate)

Version **8**

getPSNumATM( ) when passed a PostScript language font name in *psName* whose metrics are in the ATM Font Database, creates a faux font and returns its font ID in *retFondID*, regardless whether the font already exists. Application should check to see if the font exists before calling this function. The metrics of the faux font will be the same as the metrics of the font specified by *psName*.

psName Specifies the PostScript language font name of the font whose faux is being requested.

retFondID Returns the FOND ID of the faux font created.

doCreate Typically, *TRUE* will be passed in. This specifies that the application wants ATM to create the faux font.

If an application just wants to check if *psName*'s metrics are in the font database without creating the faux font, pass in *FALSE*. If *ATM_NOERR* is returned, then the font metrics are in the font database but ATM won't create a faux font at this time. *retFondID* is set to zero if *doCreate* is *FALSE*.

BUG: In ATM v. 3.5, getPSNumATM( ) returns ATM_NOERR even if the font is NOT in the database. This was fixed in v. 3.5.1.

Partial Work-Around: Check retFondID. It will be zero if ATM didn't create a faux font. Note that if doCreate is FALSE, ATM will always return a zero in retFondID, therefore, there is no work-around for getPSNumATM( ) with doCreate = FALSE.

getPSNumATM( ) returns *ATM_SUBST_DISABLED* if font substitution is disabled in the ATM Control Panel. *ATM_SUBST_PROT* is returned if the user does not have a legal serial number. In addition, there are several new error codes relating to the font database. *DB_FILE_MISSING* is returned if the database does not exist. *DB_FONT_NOT_FOUND* is returned if the requested font is not in the database. In other situations, **getPSNumATM( )** returns the appropriate ATM error code. See Appendix A for more additional error codes relating to the font substitution feature.

7 Limitations

The major limitation of the interface in ATM software versions prior to version 3.0 (API version 5) is that there is no direct support for printing. If a QuickDraw picture is being recorded, then the text showing routines return without imaging any characters.

If your application prints by spooling a QuickDraw picture, you are responsible for ensuring that the proper the ATM software calls are made when your picture is played back. This is also true for print drivers that spool a QuickDraw picture as part of the printing process; it is up to them to provide a means to intercept these calls and make the calls when the rasterization actually takes place. This is also true when printing to PostScript output devices. The application must generate the correct PostScript language code to cause the proper imaging to take place.

Current ATM software supports the following QuickDraw pen modes: *srcCopy, srcOR, srcXor, srcBic, notSrcCopy,* and *hilite.* Additionally, the pen modes *patCopy, patOR, patXor, patBic,* and *notPatCopy* are mapped into the *src* equivalents, then imaged with the ATM software. If the ATM software is invoked through the normal QuickDraw interface with any pen mode besides these, QuickDraw will be used to image the text.

- If one of the unsupported pen modes is used with a call to **showTextATMErr( )** or **xyShowTextATM( )**, then the result is undefined. This is considered a bug in the current ATM software and might be fixed in future releases.

- When using **showTextATMErr( )**, **xyShowTextATM( )**, and **getOutlineATM( )** calls, no portion of a character can be outside the 32k range. For example, in the transformation matrix passed to these routines, if *tx*=31k and the character width is 5k, the ATM software can crash. This is a bug and will be fixed in future releases.

- When used through the standard QuickDraw interface, the ATM software first checks whether there is an existing bitmap font that matches the application request. For example, if an application requests 12 point Times* and the 12 point Times bitmap font is installed, the ATM software will let QuickDraw image the text. If the ATM software is invoked with a call to **showTextATMErr( )** or **xyShowTextATM( )**, there is no attempt to match an existing screen font bitmap. This might be changed in future releases.

- The ATM software supports a scaling factor in the matrix of 8000 or 255 times the size of the largest point size installed in the system, whichever is smaller.

- The fill routines do solid fills, they do not switch to the *grafPort*'s current pen pattern.

- **getOutlineATM( )** returns the path in the same order that the font designer created the paths. Therefore, an application can not depend on the outer path being returned before the inner path.

- ATM software must truncate the matrix data *tx* and *ty* to get to an integral QuickDraw coordinate. An application might add 0.5, then take the floor to get better averaging. Note that in the example code in this document, the **FIX( )** macro handles this.

- During a call to **getOutlineATM( )**, the matrix data *tx* & *ty* do not update as they do with **showTextATMErr( )** and **xyShowTextATM( )**.

Appendix A: ATM software Error Codes

A.1 ATM Software Error Codes

The following are ATM software error codes as defined in *ATM.h*. These codes are returned by the non-Boolean routines, for example, **ATMGetOutline( )** and the ATM software fill routines.

A.1.1 ATM Software Error Codes Added in Version 4

The following error codes were added in version 4 (ATM v. 2.0)

```
#define   ATM_NOERR(0)                    /* Normal return */

#define   ATM_NO_VALID_FONT(-1)           /* can't find proper outline font */

#define   ATM_CANTHAPPEN(-2)              /* Internal ATM error */

#define   ATM_BAD_MATRIX(-3)              /* Matrix undefined or matrix too big */

#define   ATM_MEMORY(-4)                  /* Ran out of memory */

#define   ATM_WRONG_VERSION(-5)           /* currently installed ATM driver doesn't support this
                                             interface */

#define   ATM_NOT_ON(-6)                  /* the ATM driver is missing or has been turned off */

#define   ATM_FILL_ORDER(-7)              /* inconsistent fill calls, e.g.() without
                                             ATMStartFill() */

#define   ATM_CANCELLED(-8)               /* the client halted an operation, e.g.:
                                             a callback from getOutlineATM returned 0 */

#define   ATM_NO_CHAR(-9)                 /* the font does not have an outline for this
                                             character code */

#define   ATM_BAD_LENGTH(-10)             /* showTextATM() or xyShowTextATM() was called with
                                             length argument <= 0 or > 255 */
```

A.1.2 ATM Software Error Codes Added in Version 5

The following error codes were added in version 5 (ATM v. 3.0).

```
/*   (-11) Not in use in procs version 5   */

#define   ATM_NOT_BLENDED_FONT(-12)        /* This font is not a blended font. */

#define   ATM_BASEDESIGN(-13)              /* This operation is not allowed on a base design
                                              (e.g. deleting FOND). */

#define   ATM_TEMPFONT_PROB(-14)           /* problem working with a temporary font. */

#define   ATM_ILL_OPER(-15)                /* Can't perform this operation on this font.*/

#define   ATM_FONTFIT_FAIL(-16)            /* FontFit() failed (also from CopyFit()!). */

#define   ATM_MISS_BASEDESIGN(-17)         /* Missing base design FOND. */

#define   ATM_NO_BLENDED_FONTS(-18)        /* no Multiple Master fonts installed */

#define   ATM_PICKER_CANCELLED(-19)        /* Picker "Cancel" button selected */

#define   ATM_CREATE_FONT_FAIL(-20)        /* general font creation failure */

#define   ATM_DISK_FULL(-21)               /* out of disk space */

#define   ATM_WRITE_PROTECTED(-22)         /* volume or file is locked */

#define   ATM_IO_ERROR(-23)                /* I/O error */
```

A.1.3 ATM Software Error Codes Added in Version 7 (Japanese version)

The following error codes were added in version 7 (ATM v. 2.0.x Japanese version).

```
#define   ATM_COPY_PROT(-24)               /* font is copy-protected */

#define   ATM_PROT_OUTLINE(-25)            /* this outline is copy-protected */
```

A.1.4 ATM Software Error Codes Added in Version 8

The following error codes were added in version 8 (ATM v. 3.5).

```
#define   ATM_SUBST_PROT(-2627)           /* font substitution is copy-protected */

#define   ATM_SUBST_DISABLED(-2628)       /* font substitution is turned off */

#define   DB_FILE_EXPIRED(-2500)          /* Database has expired (beta only) */

#define   DB_FILE_DAMAGED(-2501)          /* Database has been damaged */

#define   DB_FILE_MISSING(-2502)          /* Font database missing */

#define   DB_FILE_BUSY(-2503)             /* Font database already in use */

#define   DB_OUT_OF_MEMORY(-2504)         /* Not enough memory for task */

#define   DB_FONT_NOT_FOUND(-2505)        /* Font not found in database */

#define   DB_BAD_REF_NUM(-2506)           /* Illegal ref_num sent to database */

#define   DB_BAD_VERSION(-2507)           /* Requested version not available */

#define   DB_NOT_A_MM_FONT(-2508)         /* Font is not Multiple Master Font */
```

Appendix B: More On Transformation Matrices

A clarification of the discussion about transformation matrices is in order for those familiar with the PostScript language or transformation matrices in general. In the PostScript language, the user space coordinate system is a standard Cartesian coordinate system with each unit exactly 1/72 inch. This is the same for all PostScript language output devices. The Current Transformation Matrix (CTM) performs the mapping of user space coordinates into device space coordinates. The device space coordinate system is that of the physical output device where one unit is equal to one device pixel and is therefore resolution-dependent. In addition, the origin of coordinates and the orientation of the coordinate system frequently differs from the user space coordinate system.

A PostScript language program requests that a transformation be applied to the user space coordinate system by using the **rotate**, **scale**, or **translate** operators. The appropriate transformation matrix is then concatenated to the CTM to create a new CTM´ that corresponds to the mapping of the new coordinate system into device space. This new CTM´ is calculated as follows:

```
CTM´ = T× CTM
```

where *T* represents the transformation of the current user space into the new user space and the multiplication is a matrix multiplication.

Consider the default CTM necessary to map one unit of user space into the QuickDraw coordinate system. Since QuickDraw is considered to have one screen pixel = 1/72″, there is no scaling necessary to map user space into device space.

One difference between user space and device space is that the QuickDraw coordinate system has y increasing down the screen, while the PostScript language coordinate system has y increasing up the screen. To account for this scaling, the y scale factor in the CTM must be –1. If we assume that the coordinate origins coincide, then the CTM should look like

```
[ 1 0 0 –1 0 0 ]
```

To transform user space into a new user space, we multiply by our transformation **T**. In the PostScript language, the matrix **T** looks like

```
T = [ Mx (Mx tan(α)) (My tan(β)) My tx ty ]
```

where *Mx* is the scale factor in *x*, *My* is the scale factor in *y* and *a* and *b* are the skew of the transformed x and y axes respectively. The coordinates *tx* and *ty* are the translation of the original origin to the new origin. Figure 1 shows this geometry.

When we concatenate **T** and the CTM we get

```
CTM´ = [ Mx (–Mx tan(α)) (My tan(β)) –My tx ty ]
```

This is identical to the matrix we specified in equation 1. This is the matrix we pass to the **showTextATMErr** and **xyShowTextATM** calls.

To calculate the matrix for rotated text, note that for uniform rotation, the angle $\beta = -\alpha$. If we write that Mx=Mcos(a) and My=Mcos(b) (M is the point size of the font we wish to rotate), then the CTM can be written as

```
CTM´ = [ Mcos(α) –Msin(α) –Msin(α) –Mcos(α) tx ty ]
```

just as indicated in equation 2.

For those interested in these transformations of coordinate spaces, see the *PostScript Language Reference Manual*, *Second Edition*. For a complete mathematical explanation of matrix operations and their uses in describing geometrical transformations, see *Principles of Interactive Computer Graphics* by W.M. Newman and R.F. Sproull, published by McGraw-Hill.

Appendix C: Converting From Pre-Version 5 API

- Variables of type *int16* are now type *short.*
- **ATMinit( )** is now **initVersionATM( ).**
- The call names changed from ATM... to ...ATM (for example, **ATMGetOutline( )** is now **getOutlineATM( )**).
- If your application used a version of *showText* that returned an error code as a parameter, the application should now use **showTextATMErr( )** or **xyShowTextATMErr( ).**
- The strings to the text showing calls are now type (*Byte **)**.**
- The parameter to **initVersionATM( )**is now a *short* (**ATMInit( )**parameter was an *int32*).
- *clientHook* in **getOutlineATM( )** and its callback parameters are now type *Ptr.*
- There are now more error codes (see Appendix A).

Appendix D: Changes Since Earlier Versions

Changes since February 14, 1992 version

- Descriptions about the function **getOutlineATM2( )** has been added. It can be used with procs version **7** which is available in ATM software Japanese version 2.0.3 (Japanese version only).

- Section 6, “SuperATM APIs” has been added. Section 6 contains descriptions about the functions **isSubstFontATM( )** and **getPSNumATM( )**.

- Parts of the document have been modified to correspond to ATM software release version 3.5 and Japanese version. Appendix A now contains additional error codes returned by new API routines.

Changes since August 13, 1991

- Converted to a new format.

- This version has been updated to correspond to the release of ATM software release version 3.0. See Appendix C for more specific information about converting your code to use the new API.

Changes since February 11, 1991

- Updated the addresses on the cover page.

Changes since February 12, 1990 version

- This version has been updated to correspond to the release of ATM software version 2.0.

- New routines corresponding to back door version 4 have been added.

- Back door routine names were updated to be consistent with the version 4 back door.

- An error code argument was added to the back door *show* routines; consequently, there were minor changes to the existing sample code.

- The conversion: `(Fixed) ( q * 65536.0 + 0.5)` in the sample code was changed to: `(Fixed) floor (q + 0.5) * 65536.0`.

Changes since October 30, 1989 version

- The phrases *Adobe Type Manager* and *ATM* were modified to read *Adobe Type Manager software* and *ATM software*.
- Various other minor typographical and formatting errors were corrected.
- The Addendum was moved to appear before the appendices.
- The part number changed from ATM0002 to LPS5072.

Index