Copyright © 1998 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated.

No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a “PostScript printing device,” “PostScript display device,” or similar item refers to a printing device, display device or item (respectively) which contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items which purport to be merely compatible.

Adobe, the Adobe logo, Acrobat, AdobePS, Adobe Type Manager, ATM, PostScript, the PostScript logo and SuperATM are trademarks of Adobe Systems Incorporated. *Palatino is a registered trademark of Linotype-Hell AG and its subsidiaries. Macintosh and Quickdraw are trademarks of Apple Computer, Incorporated, registered in the U.S. and other countries. PowerPC is a trademark of International Business Machines Corporation. Other brand or product names are the trademarks or registered trademarks of their respective holders.

THIS PUBLICATION AND THE INFORMATION HEREIN IS FURNISHED AS IS, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY ADOBE SYSTEMS INCORPORATED. ADOBE SYSTEMS INCORPORATED ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES, MAKES NO WARRANTY OF ANY KIND (EXPRESS, IMPLIED, OR STATUTORY) WITH RESPECT TO THIS PUBLICATION, AND EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSES, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS.
Contents

Examples v

Figures vii

Adobe Type Manager Software API: Macintosh 1

1 Introduction to Adobe Type Manager and the Adobe Type Manager Applications Programmer Interface 1
   Contents and Organization of this document 2

2 Application Developer's View of New ATM 4.0 Features 4
   Advanced User Features for ATM 4.0 4

3 Overview of the ATM API 7

4 The Transformation Matrix 9
   More On The Transformation Matrix 16

5 Structures, Callback Functions, and Data Types 18

6 ATM Software API Functions 20
   API Information Functions 22
   API Selection Functions 30
   API Outline/Fill Functions 35
   API TextOut Functions 45

7 ATM Return Values and Flags 47
   Return values for non-Boolean functions 48
   Return values for the font substitution database 50
   Return values for the getATMInfo function call 51
   Return values for SFNT/GX support 51

Appendix A: Changes Since Earlier Versions 51

Appendix B: ATM API Usage Table 53

Appendix C: Style Bits Positions 55

Appendix D: Obsolete ATM API Functions 57
Figures

Figure 1  The skew of the coordinate system  10
Figure 2  Upright text  11
Figure 3  Reversed text in x and y  12
Figure 4  Rotated text  13
Figure 5  Skewed text  14
Figure 6  Text using the displacement array  15
Figure 7  Path Rendering and Filling  36
Figure 8  Curves showing Bezier control points and end points  37
Examples

Example 1  Upright text  11
Example 2  Reversed text  12
Example 3  Rotated text  13
Example 4  Skewed text  14
Example 5  Text using displacement array  15
Example 6  Example use of ATM fill functions  35
Example 7  Example use of ATMGetOutline()  42
1 Introduction to Adobe Type Manager and the Adobe Type Manager Applications Programmer Interface

The Adobe Type Manager® (ATM®) software for the Macintosh® allows PostScript® language Type 1 font programs to be used transparently by single- and double-byte application programs to image text for display and printing. Additionally, ATM software can download bitmaps for printing to PostScript printers without PostScript capabilities.

The ATM Applications Programmer Interface (API) for ATM 4.0 is compatible on both the PowerPC™ and 68K Macintosh platforms and the API is available from the Adobe Developers Association (ADA). The files can be compiled using either the Macintosh Programmers Workshop (MPW) or Metrowerks CodeWarrior®. The API is compatible with all previous versions of ATM for single-byte fonts and with ATM 3.8 or greater for double-byte fonts.

Note Versions of the ATM and the API have been released to developers, without documentation, since this document was last published and released with an API in 1993. This revision of this document (May 1998) describes both the documented and undocumented calls of previous releases, as well as information on new API calls.

Using the ATM 4.0 API, an application can perform the following functions:

- enumerate fonts
- check font availability
- request a character’s path
- get the Macintosh font menu name or PostScript font name
- override default actions
- check internal operating flags
The ATM API can also access a character’s outline data, apply a transformation, fill the resulting outline, and print the character. The API has functions for showing text using the baseline of the text.

There are two versions of the ATM 4.0 software: the standard and the Deluxe version. The system component of each is identical and therefore the same API is used with it. The standard version of ATM 4.0 supports all the features of earlier versions of ATM, plus the ability to perform on-screen font smoothing, also known as anti-aliasing of text. The standard version of ATM is bundled with products and is included in the Adobe® Acrobat® software.

ATM 4.0 Deluxe gives the user additional new features for font management including the ability to group Type 1 and TrueType fonts into sets, activate (make available) and deactivate (make unavailable) sets and fonts on an as-needed basis, export font sets, view and print font samples, and perform font substitution. In addition, ATM 4.0 Deluxe supports automatic font activation and font smoothing. These functions are controlled by the user through the control panel, and not through the API. These new features are discussed in Section 2. For complete information on the features of the ATM product, see the User’s Guide.

1.1 Contents and Organization of this document

This document covers information an application programmer should understand about the ATM software and using the ATM 4.0 API. Section 2 covers the functionality added to ATM 4.0. Section 3 covers some of the changes and additions to the ATM Applications Programmer Interface (API) since this document was last updated. The functions that make up the ATM API, and their associated data structures and return flags, are covered in detail in Sections 5, 6, and 7.

Information on transformation matrices used in the ATM API functions, and their relationship to the PostScript language Current Transformation Matrix (CTM), is covered in Section 4.

Appendix A describes the changes to this document since its inception.

Appendix B lists the ATM API functions in a usage table.

Appendix C describes the Style Bits and their associated values.

Appendix D describes the ATM API functions that are no longer recommended for use or are obsolete.
This document also lists the API functions for multiple master fonts that are described in Technical Note #5074, “Adobe Type Manager Software API for Multiple Master Fonts: Macintosh.” That document may provide additional information. For more information on multiple master fonts, see Technical Note #5015, “Type 1 Font Format Supplement.”
2 Application Developer’s View of New ATM 4.0 Features

ATM 4.0 Deluxe for the Macintosh introduces new features and added functionality.

- Auto-activation of a font on a document level
- Font substitution with the help of a metrics database
- On-screen font smoothing or anti-aliasing of text
- Font management

In the Deluxe version of ATM 4.0, the user can set these features on or off through the advanced preferences menu in the control panel. In the standard version of ATM 4.0, which is bundled with many application programs, the user can only set the font smoothing feature on or off through the control panel of ATM.

This section discusses the new control panel features which do not require API calls.

Note Currently, an application cannot query ATM to see whether a control panel feature is enabled or disabled by the user.

2.1 Advanced User Features for ATM 4.0

The auto-activation feature activates fonts which are installed in the Macintosh system but not currently active. When a document referring to an inactive font is opened in any application, the font will be activated by ATM.

There are two types of auto-activation: global and private. In global auto-activation, the font that is not active but requested by an application is placed after the system in the resource chain and is made available to the requesting application and to any other application that is currently running. When the requesting application is closed, the font remains activated. Private auto-activation means that the auto-activated font is placed before the system in the resource chain, and only the application that requested the font can use it. Another application will not “see” that font unless it requests it. In this case the font is automatically activated for the new application. When the application is closed, the font is no longer available to the system.

If a document includes a multiple master font that is available on the user’s system but is not active, and the multiple master instance that is being requested exists, ATM will auto-activate the multiple master font. If the multiple master instance does not exist, the application must create it using the appropriate API call.
*Font substitution* is the action of replacing the text of an uninstalled font within a document with a simulated font having similar characteristics. The substitution of the font is done with the appropriate font metrics so that line breaks and formatting are not compromised. ATM 4.0 Deluxe can approximate the look of missing fonts by using a sans serif or serif multiple master font built specifically for substitution and the metric information contained in the ATM font substitution database. These fonts are called Adobe Sans and Adobe Serif but do not appear within an application’s font menu. Font substitution is an updated feature for ATM previously available only with SuperATM® software.

*Note* Only fonts that are described in the metrics database can be substituted.

The *font smoothing* (anti-aliasing) functionality in ATM 4.0 is designed to rasterize and display characters in a grayscale mode to achieve a truer on-screen glyph shapes. Font smoothing is only implemented for the QuickDraw environment. It is supported for black text on a white background and for color text on a white or color background. Font smoothing works for 16- and 24-bit color devices, and for 4-, 8-, 16-, and 24-bit grayscale devices. If the destination grafport (QuickDraw’s equivalent to “graphic state” of PostScript) is a color or grayscale device, text drawing APIs will smooth the text.

If an application draws text off screen as a monochrome bitmap, ATM cannot smooth the text. But, ATM can smooth rotated text and color text on color backgrounds at all point sizes. At smaller text sizes, ATM darkens the gray shades to match the weight of the true outline of the font.

*Note* At this time, there is no implementation in the ATM API to turn font substitution on or off, font auto-activation, or anti-aliasing from an application. The user may turn these features on or off on a system-wide basis within the user interface through the advanced preferences menu for ATM 4.0 Deluxe. For ATM 4.0 standard, the user may turn the font smoothing feature on or off through the Macintosh control panel.
Finally, updating and maintaining the font menu is the responsibility of the application. The ATM software has made this function easier by providing an integer value that changes every time the font state changes. This value is the \textit{SCoc resource}, which is a 16-bit unsigned integer value that is maintained by the software. This \textit{SCoc resource} is available after the ATM software is activated; and the \textit{SCoc} value will increment every time there is a change in the font state. These changes can be caused by any one of the following events:

- Adding fonts to the System’s Fonts folder.
- Automatically activating fonts via the ATM software.
- Manually activating fonts via the ATM software.
- Deactivating fonts via the ATM software.
- Creating a substitution font via the font substitution feature of ATM Deluxe.

When the value in the \textit{SCoc resource} changes, the application should proceed to update its font menu with the appropriate Macintosh function calls.

\textit{Note} For more information on the \textit{SCoc resource}, please see Technical Note #5137, “Adobe Type Manager and Macintosh Application Compatibility Issues.”
3 Overview of the ATM API

ATM 4.0 for the Macintosh is the latest version of the font utility that rasterizes Type 1 fonts to both the screen and printer. ATM 4.0 supports both single-byte and double-byte fonts, as did version 3.9; and the ATM 4.0 API is also supported on both the PowerPC™ and 68K Macintosh platforms. On the PowerPC™ platform, special consideration is necessary for the following three (3) functions: getGlyphOutlineATM(), getOutlineATM() and getOutlineATM2().

The API interface files (ATMInterface.h and ATMInterface.c) that support these functions also support the older procsets and will work with earlier versions of the ATM software. Note, however, that the API interface files have slightly different function names than the previous interface file versions.

The API has nine (9) new functions that are briefly introduced here. Complete information on all of the functions and their associated data structures and flags are located in Sections 5, 6, and 7 of this document.

getATMInfoATM() returns information about the ATM software to an application such as whether font substitution is enabled.

g.getFontInfoATM() returns information about the outline font associated with the specified Fond ID and style in the ATMFontInfo record.

g GlyphOutlineATM() retrieves the PostScript definition of the character outline in a CID-keyed font with its character identifier specified by the glyphIndex.

g OutlineATM2() is the double-byte equivalent to getOutlineATM().

g PSNumATM() is used to create a substitution font when passed a PostScript font name.

g RunsATM() retrieves the run-encoded compressed form of a single-byte character bitmap.

g Runs2ATM() is the double-byte equivalent of getRunsATM().

showGlyphATM() shows a glyph of a CID-keyed font with its character identifiers specified by the glyphIndex.

xyshowGlyphTextATMErr() report the number of glyphs of a CID-keyed font with character identifiers specified by an array of glyphs.
The following four (4) function calls were made obsolete with the release of ATM 4.0 for the Macintosh: `showTextATM()`, `xyshowTextATM()`, `normToUserCoordsATM()`, and `userToNormCoordsATM()`.
4 The Transformation 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. The matrix structure looks like this:

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

This matrix maps the coordinate system used by the application 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:

\[
\begin{align*}
    a &= M_x \\
    b &= -M_x \tan(\alpha) \\
    c &= M_y \tan(\beta) \\
    d &= -M_y
\end{align*}
\]

The elements \( a \) and \( d \) are the horizontal and vertical scaling factors. \( M_x \) is the font size in the \( x \) direction. If \( M_x \) is 12 then the \( x \) dimension of the font will be that of a 12-point font. \( M_y \) is the font size in the \( y \) direction. The \( y \) dimension of the font will be that of an \( M_y \) 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 \( \alpha \) is the upward skew of the transformed \( x \) axis relative to the normal \( x \) axis, then \( b = -M_x \tan(\alpha) \). Similarly, if \( \beta \) is the clockwise vertical skew of the transformed \( y \) axis relative to the normal \( y \) axis, then \( c = M_y \tan(\beta) \).

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 \( \theta \) 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 \( ry \) 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 \( \alpha \) and \( \beta \). The directions of skew indicated in Figure 3 would result in positive values of both \( \alpha \) and \( \beta \).

**Examples**

The following pages contain some simple code examples showing how a matrix should look when passed into `showTextATMErr()`.
Example 1:

```c
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);
```
Example 2:

/* reversed in x. */
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0

ATMFixed Matrix *matrix;
Byte *text;
short length;
short errorCode;

/*
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) 0.0;
m.c = (ATMFixed) 0.0;
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, &errorCode);
Example 3:

```c
#define DEGTORAD (2*3.14159/360)
/* 2pi/360 */
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0
ATMFixedMatrix *matrix;
Byte *text;
short length;
short errorCode;

/*
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, &errorCode);
```
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 *matrix;
Byte *text;
short length;
short errorCode;
/*
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, &errorCode);
Example 5:

/* Chars distributed on a line of slope 1 with constant widths */
#define FIX(q) (ATMFixed) floor (q + 0.5) * 65536.0
ATMFixedMatrix *matrix;
Byte *text;
short length;
short errorCode;
/* 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, &errorCode);
4.1 More On The Transformation Matrix

A clarification about transformation matrices is included here 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 $\text{CTM}^\prime$ (CTM prime) that corresponds to the mapping of the new coordinate system into device space. This new $\text{CTM}^\prime$ is calculated as follows:

$$\text{CTM}^\prime = T \times \text{CTM}$$

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 inch, 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

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

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

$$T = \begin{bmatrix} Mx & (Mx \tan(a)) & (My \tan(b)) & My & tx & ty \end{bmatrix}$$

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. Example 1 shows this geometry.
When we concatenate $T$ and the $CTM$ we get

$$CTM' = [Mx \ (-Mx \ tan(a)) \ (My \ tan(b)) \ -My \ tx \ ty]$$

This is identical to the matrix we specified in example 1. This is the matrix passed to the function call `showTextATMErr()`.

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

$$CTM' = [M\cos(a) \ -M\sin(a) \ -M\sin(a) \ -M\cos(a) \ tx \ ty]$$

just as indicated in example 2.

5 Structures and Data Types

The following structures and data types are used by the functions defined in the ATM 4.0 API. They are supported in both Macintosh’s 68K and PowerPC™ and for both single- and double-byte fonts.

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

ATMFixedPoint typedef struct
{
    ATMFixed x, y;
} ATMFixedPoint, *ATMPFixedPoint;

ATMDevInterval typedef struct
{
    short l;
    short g;
} ATMDevInterval;

ATMDevBounds typedef struct
{
    ATMDevInterval x;
    ATMDevInterval y;
} ATMDevBounds;

ATMDevRun typedef struct
{
    ATMDevBounds bounds;
    short datalen;
    short *data;
    short *indx;
} ATMDevRun, *ATMPDevRun;

ATMRunRec typedef struct
{
    ATMDevRun run;
    long xOffset;
    long yOffset;
    unsigned long dataSize;
} ATMRunRec, *ATMPRunRec;

ATMInfoTag typedef struct
{
    unsigned longtag;
    long length;
    long tagValue;
} ATMInfoTag;
**ATMInfo**

typedef struct
{
    long tagCount;
    ATMInfoTag *tags;
} ATMInfo;

**ATMFontInfo**

typedef struct
{
    unsigned short version;
    unsigned short format;
    unsigned long formatVersion;
    unsigned short container;
    unsigned short numGlyphs;
    unsigned long flags;
    unsigned long reserved[16];
} ATMFontInfo;
6 ATM Software API Functions

The ATM 4.0 API functions are presented in four sections. A list of API functions specific to multiple master fonts are listed for completeness but not documented in this technical note.

- Section 6.1, API Information Functions, comprises the following functions:

  \[ \begin{align*}
  \text{fontAvailableATM} & \quad \text{flushCacheATM} \\
  \text{getATMInfoATM} & \quad \text{getAxisBlendInfoATM} \\
  \text{getBlendedFontTypeATM} & \quad \text{getBlessedFontNameATM} \\
  \text{getFontFamilyFONDATM} & \quad \text{getFontSpecsATM} \\
  \text{getMasterFONDATM} & \quad \text{getNumAxesATM} \\
  \text{getNumBlessedFontsATM} & \quad \text{getNumMastersATM} \\
  \text{getPSNumATM} & \quad \text{getRegularBlessFontATM} \\
  \text{getRunsATM} & \quad \text{getRuns2ATM} \\
  \text{getSubstFontInfoATM} & \quad \text{initNameAndVersionATM} \\
  \text{initVersionATM} & \quad \text{isSubstFontATM}
  \end{align*} \]

- Section 6.2, API Selection Functions, comprises the following functions:

  \[ \begin{align*}
  \text{addMacStyleToCoordsATM} & \quad \text{convertCoordsToBlendATM} \\
  \text{copyFitATM} & \quad \text{createPermBlendedFontATM} \\
  \text{createTempBlendedFontATM} & \quad \text{decodeBlendedFontNameATM} \\
  \text{fontFitATM} & \quad \text{disposeTempBlendedFontATM} \\
  \text{encodeBlendedFontNameATM} & \quad \text{disposePermBlendedFontATM} \\
  \text{MMFontPickerATM} & \quad \text{getTempBlendedFontFileIDATM}
  \end{align*} \]
• Section 6.3, API Outline/Fill Functions, comprises the following functions:

- endFillATM
- fillClosePathATM
- fillCurveToATM
- fillLineToATM
- fillMoveToATM
- getFontInfoATM
- getGlyphOutlineATM
- getOutlineATM
- getOutlineATM2
- startFillATM

• Section 6.4, API TextOut Functions, comprises the following functions:

- showGlyphATM
- showTextATM
- showTextDesignATM
- xyshowGlyphTextATMErr
- xyshowTextATMErr

**Note** Following is a list of multiple master API functions. They are added for completeness and some are defined in this document. For more information about these functions, please refer to Technical Note #5074 “Adobe Type Manager Software API With Multiple Master Fonts: Macintosh.”

- addMacStyleToCoordsATM
- convertCoordsToBlendATM
- copyFitATM
- createPermBlendedFontATM
- createTempBlendedFontATM
- decodeBlendedFontNameATM
- getAxisBlendInfoATM
- disposeTempBlendedFontATM
- fontFitATM
- encodeBlendedFontNameATMErr
- flushCacheATM
- disposePermBlendedFontATM
- getBlendedFontTypeATM
- getBlessedFontNameATM
- getFontFamilyFONDATM
- getFontSpecsATM
- getMasterFONDATM
- getNumBlessedFontsATM
- getNumAxesATM
- getNumMastersATM
- getRegularBlesssedFontATM
- getTempBlendedFontFileIDATM
- MMFontPickerATM
- showTextDesignATM
6.1 ATM API Information Functions

The functions described in this section are used to initialize the ATM software, determine the version of ATM software that is available, determine which fonts are available to the application, and perform other common ATM operations.

**Note** The term *short* refers to the returned value of a function that is an integer value of either -1, 0, or 1.

**Note** The term *blended* refers to an instance of the multiple master font. An instance is a specific interpolation of a multiple master font, where the contribution of each master design is specified by the specific array values. See Technical Note #5015, “Type 1 Font Format Supplement” for more information.

**Note** The term *blessed* refers to a multiple master font instance which is preselected by the font designer and represents the regular face in the multiple master font. See Technical Note #5015, “Type 1 Font Format Supplement” for more information.

**fontAvailableATM**

**fontAvailableATM**(short family, short style)

**fontAvailableATM()** checks to see whether the ATM software will be able to image a given typeface.

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

*style* is the QuickDraw style number. The *style bits* position values are listed in Appendix C of this document.

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

The font family number and style are passed in and the style mapping table of the FOND resource is referenced to determine which PostScript language fonts ATM can use to image the 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.

**Note** This call does not parse the file to determine whether the font is a valid Type 1 font that the ATM software can use; rather it checks only for the existence of the file.
flushCacheATM  ATMErr flushCacheATM(void)

flushCacheATM() flushes the ATM font cache.

This is basically the counterpart for the System 7 function \texttt{FlushFonts()}.

getATMInfoATM  ATMErr getATMInfoATM(ATMInfo* request)

getATMInfoATM() returns information about ATM to an application such as whether font substitution is enabled. The application fills in the tag field of the request argument. The function fills in the byte length and value. If the byte length is more than 4 bytes, the value is a pointer to the data. If it is less than 5 bytes, the value is passed the tagValue field.

\textit{Note}  \textit{Pointers are to data areas inside ATM - do not change them. All strings are Pascal strings.}

Here is a list of recognized tags:

\textit{NAME} is the name of the ATM user and it returns a pointer to a Pascal string.

\textit{ORG} is the name of the organization and it returns a pointer to a Pascal string.

\textit{SUBS} returns a Boolean variable; if \texttt{True}, then ATM and font substitution are set to \textit{on} and it is functional.

\textit{LINE} returns a Boolean variable; if \texttt{True}, then the line height is preserved, otherwise the character spacing is preserved.

\textit{OFF} returns a boolean variable; if \texttt{True}, then ATM was turned \textit{off} either by the control panel or by an internal error.

\textit{DBYT} returns a Boolean variable; if \texttt{True}, then double-byte functionality is available.
getAxisBlendInfoATM

ATMErr getAxisBlendInfoATM(short axis, short *userMin, short *userMax, StringPtr axisType, StringPtr axisLabel, StringPtr axisShortLabel)

getAxisBlendInfoATM() retrieves the axis information for the multiple master font specified as the current font and current axis ID.

userMin, userMax is the lowest and highest values allowed on this axis.

axisType is the type of axis. Examples are Weight for the axis modifying character outline weight and Width for the axis modifying character outline width.

axisLabel is the label of the axis (eg. Weight).

numSubranges shows how many labels for subranges there are (ie. Light, Medium, Bold).

Note Using the value NULL for any of these parameters means that parameter is ignored. This is useful so that the function does not return unnecessary values.

getBlendedFontTypeATM

ATMErr getBlendedFontTypeATM(StringPtr fontName, short FondID)

getBlendedFontTypeATM() checks to see if the specified font is a multiple master font.

fontName is used if the return value is not NULL, otherwise it takes the FondID.

getBlendedFontTypeATM() returns True if the specified font is a multiple master font. Otherwise, one of the ATM error code value is returned (see section 7).

getBlessedFontNameATM

ATMErr getBlessedFontNameATM(short i, StringPtr blessedFontName, Fixed *coords)

getBlessedFontNameATM() returns the i instance of a multiple master font name and its coordinates.

getFontFamilyFONDATM

ATMErr getFontFamilyFONDATM(StringPtr familyName, short *retFondID)

g.getFontFamilyFONDATM() returns the FondID given the familyName.

familyName is the PostScript name of the multiple master font.
getFontSpecsATM  ATMErr getFontSpecsATM(FontSpecs *specs)

getFontSpecsATM() returns the specifications about the current font.

This information can be used with fontFitATM().

Note  For more information on the FontSpecs structure, see the information on getFontSpecsATM() in the Technical Note #5074 “Adobe Type Manager Software API for Multiple Master Fonts: Macintosh.”

getMastersFONDATM  ATMErr getMasterFONDATM(short i, short *masterFOND)

getMastersFONDATM() checks for the i base design.

getMastersFONDATM() returns the FondID for the i base design for the given multiple master font which is specified as the current font and style in the current grafport. Otherwise, one of the ATM error code value is returned (see section 7).

This return value can be used for the CopyFit() function.

getNumAxesATM  ATMErr getNumAxesATM(short *numAxes)

getNumAxesATM() checks for the number of axes of a multiple master font.

getNumAxesATM() returns the number of axes from the given family name and axis identifier. Otherwise, an ATM error code value is returned (see section 7).

g.getNumBlessedFontsATM  ATMErr getNumBlessedFontsATM(short *numBlessedFonts)

getNumBlessedFontsATM() returns the number of primary multiple master font instances that exist.

g.getNumMastersATM  ATMErr getNumMastersATM(short *numMasters)

getNumMastersATM() returns the number of base designs of the current multiple master font. Otherwise, one of the ATM error code value is returned (see section 7).

This return value can be used for the CopyFit() function.
**getPSNumATM**

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

getPSNumATM() creates a substitute font and returns its FondID in retFondID, regardless of whether the font already exists.

*psName* specifies the PostScript language font name of the substitute font that is being requested.

*retFondID* returns the FondID of the substitute font created.

*doCreate* is usually passed the value True. This Boolean specifies that the application wants ATM to create a substitute font.

When an application is checking to see whether a PostScript font’s metrics are included in the database without creating a substitute font, *doCreate* should be passed False. If ATM_NOERR is returned, then the font metrics are in the font database but ATM won’t create a substitute font at this time. *retFondID* is set to zero if *doCreate* is False.

getPSNumATM() returns ATM_SUBST_DISABLED if font substitution is disabled in the ATM Control Panel.

getPSNumATM() returns DB_FILE_MISSING if the database does not exist.

getPSNumATM() returns DB_FONT_NOT_FOUND if the requested font’s metrics are not listed in the database. Otherwise, one of the other ATM error code values is returned (see section 7).

**getRegularBlessedFontATM**

ATMErr getRegularBlessedFontATM(short *regularID)

getRegularBlessedFontATM() finds the primary multiple master font instance currently selected in the current grafport.

getRegularBlessedFontATM() returns the index of the multiple master instance in *regularID* which is to be used as input to getBlessedFontNameATM().
getRunsATM  ATMErr getRunsATM(short c, ATMFixedMatrix *matrix, Ptr clientHook, ATMPRunRec pRunRec)

getRunsATM() returns the run-length encoded compressed form of a character bitmap for a character code pointed to by c. The character is a single-byte character in the current font.

Matrix maps one point in character space to the device space.

The calling application should set pRunRec -> run.data to point to storage and set pRunRec -> dataSize to the size of this storage in bytes for the runs format.

The runs format is the boundary in ATMDevRuns which specifies the bounding box of the bitmap. The data in the ATMDevRuns structure points to an array of short values. The point values on each row are stored end-to-end. Each row of the bitmap is represented as [count, start-x1, end-x1,...,start-xn, end-xn], where count is the number of start-xi and end-xi pairs. Each start-xi and end-xi pair represents a segment of consecutive bits which are turned on in one row of rasterized bitmaps. For each of the y coordinates between bounds y.l and bounds y.j, a row of data must exist in the array. This way, a blank row of bitmap data is represented as one zero (0) short value (i.e., count=0). A pair or x values are relative to bounds x.l. The whole bitmap may be offset by x-offset and y-offset.

getRunsATM() returns ATM_NOERR if successful, otherwise one of the ATM error code values is returned (see section 7).

getRuns2ATM  ATMErr getRuns2ATM(char *cp, ATMFixedMatrix *matrix, Ptr clientHook, ATMPRunRec pRunRec)

getRuns2ATM() has the same functionality as getRunsATM(), but accepts double-byte character bitmaps.

This function returns the run-length encoded character bitmap for a character code pointed to by cp. The character may be a single-byte or double-byte character depending on the character code and the script of the current font.
**getSubstFontInfoATM**

ATMErr getSubstFontInfoATM(StringPtr fontName, short *FondID, Boolean *found, Boolean *deferred)

getSubstFontInfoATM() returns True if the specified font is a substituted font.

If the fontName has zero length, look up is done via the FondID and returns the font name.

If the font name length is non-zero, look up is done via the font name and returns the FondID.

If the substituted font is reserved but it has not been created yet or has been flushed out, True is returned in the deferred parameter.

**initNameAndVersionATM**

short initNameAndVersionATM(short version, StringPtr name, short *errorCode)

initNameAndVersionATM() checks to see whether the named ATM software is running and if it is, then the API functions are initialized.

version is the API version number being requested.

name is usually set to a Pascal string, i.e., .ATM

initNameAndVersionATM() returns the value one (1) if the named ATM software is running and the desired version of the API routines is available. Otherwise, a value zero (0) is returned.

*Note* This is a utility function used by initVersionATM(). Do not call it directly.

**initVersionATM**

short initVersionATM (short version)

initVersionATM() checks to see if the ATM software is running and if it is, initializes the ATM API.

initVersionATM() returns the value one (1) if and only if that version of the ATM API is available. Otherwise, a value of zero (0) is returned.
isSubstFontATM    Boolean isSubstFontATM(StringPtr fontName, short FondID, short style, FontSpecs ***fontSpecs, Handle *chamName)

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

fontName specifies the Macintosh font name that is being queried. If fontName is NULL, then FondID is used.

FondID specifies the FondID of the font that is being queried. This is used only if fontName is NULL.

style specifies the Macintosh style of the font being queried. The style bits position values are listed in Appendix C of this document.

If isSubstFontATM() is True and FontSpecs is not NULL, then fontSpecs returns a handle to the fontSpecs record for the file. The calling application is responsible for disposing of this handle when it is finished.

If isSubstFontATM() is True and chamName is not NULL, then chamName returns a handle containing the PostScript language font name of the substitute font. The calling application is responsible for disposing of this handle when it is finished.

isSubstFontATM() returns True if the font is a substitute font, Otherwise, False is returned.

Note    For more information on the FontSpecs structure, see the information on getFontSpecsATM() in Technical Note #5074 “Adobe Type Manager Software API for Multiple Master Fonts: Macintosh”.
6.2 API Selection Functions

The functions described in this section are used to create and remove selected fonts.

**addMacStyleToCoordsATM**

ATMErr addMacStyleToCoordsATM(Fixed *coords, short macStyle, Fixed *newCoords, short *stylesLeft)

**addMacStyleToCoordsATM()** returns the new coordinates for any given Macintosh style for a multiple master font.

If the **bold** style bit is set to **on** for a multiple master font, a calculated amount may be added to the weight axis to embolden the font outline.

*Note* Note that a font designer does not have the choice to modify the coordinates for the **Bold**, **Condensed** or **Expanded** bits, so these coordinates will return unchanged. Whatever styles are not part of the font design are returned in **stylesLeft**. These styles might be handled by QuickDraw (like **Outline**, **Shadow**, etc).

**convertCoordsToBlendATM**

ATMErr convertCoordsToBlendATM(Fixed *coords, Fixed *weightVector)

**convertCoordsToBlendATM()** converts the design coordinates of the current font to a blend vector. This is necessary for applications that output PostScript directly - the return value is suitable for passing to the PostScript operator **makeBlendedFont**.
**copyFitATM**

`ATMErr copyFitATM(short method, Fixed TargetWidth, Fixed *beginDesign, Fixed *baseWidths, Fixed *resultWidth, Fixed *resultDesign)`

`copyFitATM()` is used to select a multiple master font instance that will match the given width of a string.

`baseWidths` is the array with the width of the string for each one of the base designs.

`targetWidth` is the destination string array width.

`beginDesign` is the original coordinates.

ATM will return the best font instance it can find in the `resultDesign` parameter and the actual width of the string in `resultWidth`.

**Note** This is not supported in ATM 3.9. **ATM_NOT_BLENDED_FONT** is the error that is returned.

**createPermBlendedFontATM**

`ATMErr createPermBlendedFontATM(StringPtr fontName, short fontSize, short fontFileID, short *retFondID)`

`createPermBlendedFontATM()` creates a permanent multiple master font instance with a bitmapped font of a given size for the given outline file. Currently the zero-sized case is the only one supported, which will create a thumbnail 10-point bitmapped font file for use in the application.

The fonts should be disposed of with the function call `disposePermBlendedFontATM()`.

**createTempBlendedFontATM**

`ATMErr createTempBlendedFontATM(short numAxes, Fixed *coords, short *useFondID)`

`createTempBlendedFontATM()` creates a temporary multiple master font instance for the user interface. When this function is first called, `useFondID` is initialized to zero and a new `FondID` is returned. When multiple instances are displayed, new `FondIDs` are returned in `useFondID` until the display is completed.

The fonts should be disposed of with the function call `disposeTempBlendedFontATM()`.
**decodeBlendedFontNameATM**  
ATMErr decodeBlendedFontNameATM(StringPtr blendName, StringPtr familyName, short *numAxes, Fixed *coords, StringPtr displayInstanceStr)

**decodeBlendedFontNameATM()** returns the family name for a font, the number of axes in the font, the coordinates within those axes, and the display instance string when a multiple master font instance name is provided.

*blendName* is a multiple master font instance name to be decoded. For example, MyriaMM\_500 wt 120 wd.

**decodeBlendedFontNameATM()** returns the string value in *familyName*. It is the information before the underscore in a multiple master instance name. For example, MyriaMM.

**decodeBlendedFontNameATM()** returns the value in *numAxes*. This is the number of axes in the font.

**decodeBlendedFontNameATM()** returns the value in *coords* which is an array of *Fixed* values that represent the fonts coordinates in user space (range userMin – userMax) for each axis. This can be used with any of the API calls that require *user coords* as input parameters.

**decodeBlendedFontNameATM()** returns the string value in *displayInstanceStr*. This is the value after the underscore. For example, the multiple master font instance name is MyriaMM\_500 wt 120 wd, the return value for *displayInstanceStr* would be 500 wt 120 wd.

Any output parameters can be NULL, in which case the values are ignored.

**Note** An application can use this function with fonts created by **createTempBlendedFontATM()**, in which case, *displayInstanceStr* is ignored.

Because **decodeBlendedFontNameATM()** parses the *blendName* string and breaks it into sub-strings, it provides a simple method of obtaining a family name from a full PostScript font name.

**disposePermBlendedFontATM**  
ATMErr createPermBlendedFontATM(short FondID)

Disposes of the permanent font created by **createPermBlendedFontATM()**.

**disposeTempBlendedFontATM**  
ATMErr disposeTempBlendedFontATM(short FondID)

Disposes of the temporary font created by **createTempBlendedFontATM()**.
**encodeBlendedFontNameATM**  
ATMErr encodeBlendedFontNameATM(StringPtr familyName, short numAxes, Fixed *coords, StringPtr blendName)

**encodeBlendedFontNameATM** returns a new name suitable for use with the Macintosh Font Manager and with the ATM API function calls.

*familyName* is the family name of a font. It is the information before the underscore in a multiple master font instance name. For example, MyriaMM.

*numAxes* is the number of axes in the multiple master font.

*coords* is an array of Fixed values that represent the font’s coordinates in user space (range `userMin` – `userMax`) for each axis.

**encodeBlendedFontNameATM** returns the string in *blendName*. This is the multiple master font instance name that can be used with the API function calls requiring a font name string. For example, the returned *blendName* string MyriaMM_450 wt 600 wd, is an instance name from the Myriad multiple master typeface whose coordinates are 450 on the weight axis and 600 on the width axis.

*Note* Note that the display instance string may not always be delimited by spaces.

The returned string in *blendName* can be used with:

- `getBlendedFontTypeATM()`,
- `createPermBlendedFontATM()`,
- `decodeBlendedFontNameATM()`,
- `MMFontPickerATM()`,
- `GetFNum()`.

In combination with the `decodeBlendedFontNameATM()` function an application can convert from a PostScript language font name to a font instance name that can be used with API calls and `GetFNum()`. For example, convert from the font name MyriadMM_450_RG_600_NO to the instance name MyriaMM_450 RG 600 NO.

To achieve this, pass the PostScript language name as the *blendName* parameter to `decodeBlendedFontNameATM()`. Then pass the output of the decode function call into `encodeBlendedFontNameATM()`. The resulting *blendName* will be the correct format to use with the API functions and `GetFNum()`. 
fontFitATM  ATMErr fontFitATM(Fixed *origCoords, short numTargets, short *varyAxes, Fixed *targetMetrics, Fixed **masterMetrics, Fixed *retCoords, Fixed *retWeightVector)

fontFitATM() is the primary function used with copyfitting and other multiple master operations.

fontFitATM() returns its results in retCoords and retWeightVector -if either argument is NULL, it is ignored.

numTargets is passed the integer value 2, one for string length and one for stem width.

targetMetrics takes 2 arguments. The first is the desired length of the text (ie., the width of the column) and the second specifies the stem width.

masterMetrics takes 2 arguments, the first is a pointer to the widths of the strings at each master design (loop through the masters using getMasterFONDATM()). And the second entry in masterMetrics is the stem widths for each master design.

Note This function is not supported in ATM 3.9, ATM_NOT_BLENDED_FONT is the error returned.

getTempBlendedFontFileIDATM ATMErr getTempBlendedFontFileIDATM(short *fileID)

getTempBlendedFontFileIDATM() returns the file ID of the temporary file in which ATM stores newly created fonts.

MMFontPickerATM  ATMErr MMFontPickerATM(MMFP_Parms *parms, MMFP_Reply *reply)

MMFontPickerATM() is the multiple master Font Creator interface.

MMFontPickerATM() displays a dialog allowing the user to navigate through the design space of any installed multiple master font. The user is allowed to create an instance within that font’s design space.

MMFontPickerATM() returns information about the user’s selection, if any was made.
6.3 API Outline/Fill Functions

Using `getOutlineATM()` and its associated functions, an application has the capability to get the character outline data. The outline is returned as a series of points in device space. These points can then be manipulated by the application. `getOutlineATM()` works similarly to the PostScript language operator `pathforall`.

`endFillATM`  
short `endFillATM(void)`

`endFillATM()` is used to paint the area enclosed by the current path (the path drawn since the last `startFillATM()` with the foreground color implicit in the current `grafPort`. The inside of the current path is determined by the PostScript language even-odd winding rule. See the PostScript Language Reference Manual, Second Edition for details.

**Note** Note that the QuickDraw function `ForeColor` may be changed before `endFillATM()` is executed.

`endFillATM()` returns `ATM_NOERR` if the call was successful, otherwise one of the ATM error code values is returned (see section 7).

**Example 6:**

```c
#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();
```
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.

**fillClosepathATM**  
short `fillClosePathATM(void)`

`fillClosePathATM()` is used to close the current subpath by appending a line segment connecting the current point to the subpath’s starting point.

`fillClosePathATM()` returns `ATM_NOERR` if the call was successful, otherwise one of the ATM error code values is returned (see section 7).
fillCurveToATM short fillCurveToATM(ATMFixedPoint pc1, ATMFixedPoint pc2, ATMFixedPoint pc3)

fillCurveToATM() adds a Bézier curve segment to the current path. It takes three pointers to ATMFixedPoint, which represent the locations \((x_1,y_1), (x_2,y_2), (x_3,y_3)\) in the current grafPort, and adds a Bézier curve segment to the current path between the current point, referred to here as the point \((x_0, y_0)\) and the point \((x_3, y_3)\), using \((x_1, y_1)\) and \((x_2, y_2)\) as the Bézier 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 one of the ATM error code values is returned (see section 7).

**Figure 8** 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:

\[
x(t) = a_x t^3 + b_x t^2 + c_x t + x_0 \\
y(t) = a_y t^3 + b_y t^2 + c_y t + y_0
\]

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

\[
x_1 = x_0 + c_x /3 \\
x_2 = x_1 + (c_x + b_x) /3 \\
x_3 = x_0 + c_x + b_x + a_x
\]

\[
y_1 = y_0 + c_y /3 \\
y_2 = y_1 + (c_y + b_y) /3 \\
y_3 = y_0 + c_y + b_y + a_y
\]
**fillLineToATM**  
short fillLineToATM(ATMFixedPoint pc)

*fillLineToATM* takes a pointer to an *ATMFixedPoint*, which represents an *x*, *y* location in the current *grafPort* and appends a straight line segment from the current point to the point (*x*, *y*) which then becomes the new current point. *fillLineToATM* is similar to the PostScript language *lineto* operator.

*fillLineToATM* returns *ATM_NOERR* if the call was successful, otherwise one of the ATM error code values is returned (see section 7).

**fillMoveToATM**  
short fillMoveToATM(ATMFixedPoint pc)

*fillMoveToATM* is used to establish a new current point without adding a line segment to the current path. *fillMoveToATM* takes a pointer to an *ATMFixedPoint*, which represents an *x*, *y* location in the current *grafPort*, then the function starts a new subpath of the current character path and sets the current point to *x*, *y*. The function must be called at least once to establish a starting point before any calls to *fillLineToATM* or *fillCurveToATM* are executed. It is similar to the PostScript language *moveto* operator.

*fillMoveToATM* returns *ATM_NOERR* if the call was successful, otherwise one of the ATM error code values is returned (see section 7).

**getFontInfoATM**  
short getFontInfoATM(short FondID, short style, ATMFontInfo *fontInfo)

*getFontInfoATM* returns information about the outline font associated with the specified *FondID* and the style in the *ATMFontInfo* record.

*getFontInfoATM* returns *ATM_NO_VALID_FONT* if it is not a Type 1 font.
**getGlyphOutlineATM**
short getGlyphOutlineATM(unsigned short glyphIndex, ATMFixedMatrix *matrix, Ptr clientHook, ATMxProto2(MoveTo, short, Ptr clientHook, ATMPFixedPoint pc), ATMxProto2(LineTo, short, Ptr clientHook, ATMPFixedPoint pc), ATMxProto4(CurveTo, short, Ptr clientHook, ATMPFixedPoint pc1, ATMPFixedPoint pc2, ATMPFixedPoint pc3), ATMxProto1(ClosePath, short, Ptr clientHook))

**getGlyphOutlineATM()** provides applications with the capability to get the PostScript language definition of a character outline for a glyph of a CID-keyed font with its character identifier specified by glyphIndex. The current font must be set to a FondID which is associated with the CID-keyed font.

The routines should return the value one (1) if the function **getGlyphOutlineATM()** is to continue for the remainder of the current character; it should return the value zero (0) if the function **getGlyphOutlineATM()** is to be terminated.

**getGlyphOutlineATM()** returns **ATM_NOERR** if the call was successful, otherwise one of the ATM error code values is returned (see section 7).

**getOutlineATM**
short getOutlineATM(short c, ATMFixedMatrix *matrix, Ptr clientHook, ATMxProto2(MoveTo, short, Ptr clientHook, ATMPFixedPoint pc), ATMxProto2(LineTo, short, Ptr clientHook, ATMPFixedPoint pc), ATMxProto4(CurveTo, short, Ptr clientHook, ATMPFixedPoint pc1, ATMPFixedPoint pc2, ATMPFixedPoint pc3), ATMxProto1(ClosePath, short, Ptr clientHook))

**getOutlineATM()** provides applications with the capability to get the 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 function calls or using the moveto, lineto, curveto, and closepath PostScript operations.

*c* is the character whose outline is to be retrieved.

**matrix** allows for the transformation (rotation, skewing, scaling, translation) of the character outline returned. This matrix is the same as the matrix used by **showTextATMErr()** and is described in section 4, “The Transformation Matrix.” When using **showTextATMErr()**, however, the current QuickDraw pen location is relevant.

In the case of **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 located at coordinates (0, 0).
For example, if the matrix has:

\[
\begin{align*}
m.a &= \text{FIX}(50.0) \\
m.b &= m.c = (\text{ATMFixed})\ 0 \\
m.d &= -m.a \\
m.tx &= \text{FIX}(100.0) \text{ and } m.ty = \text{FIX}(100.0)
\end{align*}
\]

the points returned by `getOutlineATM()` will be the QuickDraw coordinates of a regular 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 \(m.tx = m.ty = \text{FIX}(100.0)\) can be used.

However, if the outline can be displayed arbitrarily, it may be best to call `getOutlineATM()` with \(m.tx = m.ty = (\text{ATMFixed})\ 0\) and perform the necessary transformations.

`clientHook` is a 32-bit value that `getOutlineATM()` will pass to all the application supplied routines. `clientHook` is commonly used as a pointer to a list of data passed to the routines by `getOutlineATM()`.

`MoveTo`, `LineTo`, `CurveTo`, and `ClosePath` are routines that correspond to the PostScript operators (`moveto`, `lineto`, `curveto`, and `closepath`) that might be processed by `getOutlineATM()`. A return value of one (1) indicates that the function will be called again for the current character. A value of zero (0) is returned when `getOutlineATM()` is terminated.

`getOutlineATM()` returns `ATM_NOERR` if the call was successful, otherwise another ATM error code value is returned (see section 7).

`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 one. 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 in the string. The character code is interpreted the same way `DrawText()` would interpret it, and takes into account the properties of the currently selected font. That is, if the current font is Japanese, the string is interpreted as
containing Shift-JIS character codes. If the current font is Roman, the string is interpreted as having character codes of the extended ASCII character set used by the Macintosh.

A return value of one (1) indicates that the function will be called again for the current character. A value of zero (0) is returned when `getOutlineATM2()` is terminated.

`getOutlineATM2()` returns `ATM_NOERR` for a successful operation, after it has enumerated the character path. Otherwise, one of the ATM error code values is returned (see section 7).

There are specific error codes assigned for the Japanese versions of ATM software. When Chinese and Korean versions of ATM are produced and released, these error codes will also apply to those versions.

In cases where the font installed has its character outline data copy protected, `getOutlineATM2()` will return the value `ATM_PROT_OUTLINE`. Certain double-byte font products from Adobe and other vendors have protected outline data (also called *copy protected*). This function will not return outline data for characters in a protected font.

Following is an example of how to obtain a character’s outline description, sample code is also given, which should work on all Macintosh platforms. Although the example illustrates the usage of the function call `getOutlineATM()`, it is also the illustration for the function call `getOutlineATM2()`. The operator `FIX()` is used throughout the example. It is defined as follows:

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

**Example 7:**

```c
#include ...

/*
   If we are using routine descriptor (i.e. PPC), allocate the UPPs for each callback routine.
*/
#if USESROUTINEDESCRIPITORS
   UniversalProcPtr myLineToUPP, myMoveToUPP, myCurveToUPP, myClosePathUPP;
#endif
Boolean InitGetOutlineTest(void);
OSErr GetCharOutline(short, short, short);
/*
   Called once. Sets up the UPPs if we are using routine descriptors. Calls InitOutlineCallbacks() to set up the callback routines.
*/
Boolean InitGetOutlineTest(void)
{
```
#if USESROUTINEDESCRIPTORS
    myMoveToUPP = NewMoveToProc(myMoveTo);
    myLineToUPP = NewLineToProc(myLineTo);
    myCurveToUPP = NewCurveToProc(myCurveTo);
    myClosePathUPP = NewClosePathProc(myClosePath);
#endif

InitOutlineCallbacks();
return true;
}
OSErr GetCharOutline(short fontID, short fontSize, short character)
{
    Handle path_handle = NewHandle(0); /* Handle for callback storage */
    ATMFixedMatrix myMatrix;
    float PointFTranslation;
    float fRotation;
    float PointFScale;
    float fSize;
    OSErr myErr = noErr;
    if(path_handle)
    {
        TextFont(fontID);
        fRotation = 0.0; /* Expressed in degrees */
        fTranslation.x = 0.0; /* Horizontal translation from the QuickDraw origin */
        fTranslation.y = 180.0; /* Vertical translation from the QuickDraw origin */
        fScale.x = 1.0; /* Horizontal scale factor */
        fScale.y = 1.0; /* Vertical scale factor */
        fSize = 1.0 * fontSize; /* Font size in points */
        if(SetATMMatrix(&myMatrix, fSize, fScale, fRotation, fTranslation, ATMquadOne)),
        {
            #if USESROUTINEDESCRIPTORS
                myErr = getOutlineATM(character, &myMatrix, (Ptr) path_handle, (short (*)(())) myMoveToUPP, (short (*)(())) myLineToUPP, (short (*)(())) myCurveToUPP, (short (*)(())) myClosePathUPP);
            #else
                myErr = getOutlineATM(character, &myMatrix, (Ptr) path_handle, myMoveTo, myLineTo, myCurveTo, myClosePath);
            #endif
        }
    }
    else
    return ATM_BAD_MATRIX;
    DisposeHandle(path_handle);
}
else
    myErr = memFullErr;
return myErr;
A 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 operators, `getOutlineATM()` executes the corresponding application supplied routine handling the routine `clientHook` and the data that corresponds to the basic element being processed.

As shown in the parameter list of `getOutlineATM()`, all of the parameters take arguments that include `clientHook` and the appropriate number of pointer(s) to `ATMFixedPoint` points. The `moveto` and `lineto` operators each take one `ATMFixedPoint` parameter, the `curveto` gets three `ATMFixedPoint` arguments, and `closepath` requires no `ATMFixedPoint` arguments. The parameters passed to `getOutlineATM()` correspond to the procedure arguments passed to the PostScript operator `pathforall`.

The `getOutlineATM()` function returns path data in the same order that the font designer created the character. This means that an application cannot depend on the order in which character paths will be drawn. Additionally, 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. If `getOutlineATM()` requests the font style bold but the outline font is not available, the ATM software error code `ATM_NO_VALID_FONT` is returned by `getOutlineATM()`.

**Note** In most cases, the ATM software preserves the application’s 68k system registers A1 - A5 and D1 - D7 across calls to the API. But within the calls `getOutlineATM()` and `getOutlineATM2()` only register A5 is preserved across routines.

**startFillATM**

```c
short startFillATM(void)
```

`startFillATM()` places ATM software in a ready state to accept the rendering commands `fillMoveToATM()`, `fillLineToATM()`, `fillCurveToATM()` and `fillClosePathATM()`. When in this state, no changes to the QuickDraw graphics state should be made as the results are undefined.

**Note** When ATM software is in the fill state, `getOutlineATM()` or `getOutlineATM2()` should not be invoked.

`startFillATM()` returns `ATM_NOERR` if the call was successful, otherwise one of the ATM error code is returned (see section 7).
6.4 API TextOut Functions

The functions described in this section are used to show text.

**showGlyphATM**

ATMErr showGlyphATM(unsigned short glyphIndex, ATMFixedMatrix *matrix)

*showGlyphATM()* is used to show a glyph of a CID-keyed font with its character identifier specified by glyphIndex which may not need be encoded in the CMap associated with the font. The current *txFont* must be set to a *FondID* which is associated with the CID-keyed font. The glyph is transformed by the specified matrix which maps a one point character space to *device space*, relative to current pen position. The Matrix components *tx* and *ty* are updated to indicate the origin of the next glyph to be displayed.

*showGlyphATM()* returns ATM_NOERR if successful, otherwise one of the ATM error code values is returned (see section 7).

**showTextATMErr**

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

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

*text* is a pointer to an array of characters to be imaged. The characters in the array are shown relative to the current QuickDraw pen location with the orientation and scale specified by *matrix*. The font used to draw the string is selected by QuickDraw’s *TextFont()* and *TextFace()* functions.

*length* is a count of the number of characters in the string to be imaged. The length must be between 0 and 255 inclusive.

*Note* Pascal programmers should skip the first element of the string which is the *length*.

*matrix* points to a transformation matrix that can be used to rotate or skew the text. This matrix is described in section 4, “The Transformation Matrix.”

*errorCode* is set to ATM_NOERR if the call was successful. Otherwise *errorCode* is set to the appropriate ATM error code value (see section 7).

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

ATMErr showTextDesignATM(StringPtr fontFamily, Byte *text, short len, ATMFixedMatrix *matrix, Fixed *coords, Fixed *displacements, short *lenDisplayed)

This function is an extension of the set of “showText” functions which allows an application to show a text string without creating an explicit FondID.

showTextDesignATM() returns the x and y delta width values between each character if displacements is not NULL. If displacements is NULL, then it will use the widths value from the Fond or NFNT.

fontFamily should pass the parameter NULL.

coords should pass the parameter NULL unless this is a multiple master font.

showTextDesignATM() returns ATM_NOERR if successful, otherwise one of the ATM error code values is returned (see section 7).

xyshowGlyphTextATMErr  

short xyshowGlyphTextATMErr(unsigned short *glyphs, short numGlyphs, ATMFixedMatrix *matrix, Fixed *displacements, ATMErr *errorCode)

The functionality of xyshowGlyphTextATMErr() is a super-set of showGlyphATM().

xyshowGlyphTextATMErr() is used to show the number of glyphs in a CID-keyed font with character identifiers specified by a glyph array. The current txFont must be set to a FondID which is associated with a CID-keyed font. The glyph is transformed by the specified matrix which maps a one point character space to the device space, relative to current pen position. The matrix components tx and ty are updated to indicate the origin of the next glyph to be displayed.

xyshowGlyphTextATMErr() returns ATM_NOERR if successful, otherwise one of the ATM error code values is returned (see section 7).
xyshowTextATMErr: short xyshowTextATMErr(Byte *text, short length, ATMFixedMatrix *matrix, ATMFixed *displacements, short *errorCode)

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

*text* is a pointer to an array of characters to be imaged. 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 strings is selected by QuickDraw’s **TextFont()** and **TextFace()** functions.

length is a count of the number of characters in the string to be imaged. The length must be between 0 and 255 inclusive.

Note: **Pascal programmers should skip the first element of the string which is the length.**

*matrix* points to a transformation matrix that can be used to rotate or skew the text. This matrix is described in section 4, “The Transformation Matrix.”

*displacements* is a pointer to an array of fixed point numbers which are displacement values in the x and y direction for each character in the string.

The number of elements in the displacement array is twice the number of characters being imaged. This **displacements** array is a series or 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 for the first character in the string. The next element in the array is the y width for the first character, and so on. For the ith element of the array, the width used for that character is \( wx = \text{displacements}[2i] \), \( wy = \text{displacements}[2i+1] \). The numbers in the array are the widths as if the font were being imaged as a one point font. The values are appropriately scaled, rotated, or skewed by the transformation matrix passed in to the **xyshowTextATMErr()** function. For example, for most Roman typefaces, the y value will be 0, even when imaging rotated text.

*errorCode* is set to ATM_NOERR if the call is successful, otherwise, *errorCode* is set to the appropriate ATM error code value (see section 7).

**xyshowTextATMErr()** returns the number of characters in the string that were not imaged.
7 ATM Return Values and Flags

The following return values, flags, type bits, and flag bits are used by the functions defined in the ATM 4.0 API. They are supported in both the PowerPC™ and 68K Macintosh systems. Additionally, they are supported for all functions manipulating single- and double-byte fonts.

7.1 Return values for non-Boolean functions

- **ATM_NOERR** 0
  The typical return value

- **ATM_NO_VALID_FONT** -1
  Can’t find the proper outline font

- **ATM_CANTHAPPEN** -2
  An internal ATM error

- **ATM_BAD_MATRIX** -3
  An undefined inverse matrix or the matrix is too big

- **ATM_MEMORY** -4
  An out of memory error

- **ATM_WRONG_VERSION** -5
  Wrong version of ATM for this call

- **ATM_NOT_ON** -6
  The ATM driver is missing or is not currently running

- **ATM_FILL_ORDER** -7
  Inconsistent fill calls

- **ATM_CANCELLED** -8
  A client halted current operation

- **ATM_NO_CHAR** -9
  No outline for specified character code in this font

- **ATM_BAD_LENGTH** -10
  showTextATMERR() or xyshowTextATMERR() was called with length argument <= 0 or > 255

- **ATM_NOT_BLENDED_FONT** -12
  This face is not a font instance.

- **ATM_BASEDESIGN** -13
  This operation is not allowed on a multiple master base design (e.g. deleting FOND)
ATM_TEMPFONT_PROB -14
There was a problem working with the temporary font

ATM_Ill_Oper -15
The operation cannot be perform on this font

ATM_FONTFIT_FAIL -16
FontFit() or CopyFit() failed

ATM_MISS_BASEDESIGN -17
Missing base design FOND ID

ATM_NO_BLENDED_FONTS -18
No multiple master fonts installed

ATM_PICKER_CANCELLED -19
Font Creator “Cancel” button selected

ATM_CREATE_FONT_FAILED -20
General font creation failed

ATM_DISK_FULL -21
Out of disk space

ATM_WRITE_PROTECTED -22
Volume or file is locked (write protected)

ATM_IO_ERROR -23
I/O error

ATM_COPY_PROT -24
The outline font is copy-protected

ATM_PROT_OUTLINE -25
This character outline is copy-protected

ATM_RUN_MEMORY -26
ATMGetRuns() needs bigger client buffer

ATM_BAD_MMFOND_STRUCT -27
CopyOldToNewFond failed, with copy of data past end

ATM_LOADABLE -28
ATM is busy
7.2 Return values for the font substitution database

**ATM_SUBST_PROT** -2627
The substituted font is copy-protected

**ATM_SUBST_DISABLED** -2628
Font substitution is turned off

**ATM_BAD_FAUX_OUTLINE** -2629
Failed at updating substitution font file because it really was a file with the same name that wasn’t a substitute font file

**DB_FILE_EXPIRED** -2500
The font database has expired (for beta only)

**DB_FILE_DAMAGED** -2501
The font database has been damaged

**DB_FILE_MISSING** -2502
The font database is missing

**DB_FILE_BUSY** -2503
The font database is already in use

**DB_OUT_OF_MEMORY** -2504
There is not enough memory for the current task

**DB_FONT_NOT_FOUND** -2505
The requested font metrics are not found in the database

**DB_BAD_REF_NUM** -2506
An illegal ref_num is sent to the database

**DB_BAD_VERSION** -2507
The requested version of the metrics database is not available

**DB_NOT_A_MM_FONT** -2508
The font requested is not a multiple master font
7.3  Return values for the getATMInfo function call

   **ATM_INFO_UNKNOWN** -2400
   The *tag* was no recognized

   **ATM_INFO_BAD_TAG_COUNT** -2401
   The *tagCount* is less than 1

   **ATM_INFO_BAD_VALUE** -2402
   The requested value is bad; the given pointer is NULL

7.4  Return values for SFNT/GX support

   **ATM_NOTSFNT** -2403
   ATM does not recognize this font as having an *sfnt*

   **ATM_FAILED_SFNT_IO** -2404
   The I/O operation to an *sfnt* font failed
Appendix A: Changes Since Earlier Versions

Changes for ATM API SDK version 4.0

- Added `xyshowGlyphTextATMErr()`, which is a double-byte specific function which was added to ATM version 3.9.

- Added `getFontInfoATM()`, which is a subset of `getSubstFontInfoATM()` which was added to ATM version 3.92.

- Added `getSubstFontInfoATM()`, which is a new function to ATM Deluxe 4.0.

- `normToUserCoordsATM()` was obsoleted with release 3.9.

- `userToNormCoordsATM()` was obsoleted with release 3.9.

Some changes for release 3.8.

- Added Mixed Mode Manager support.

- Removed definitions for compiling with THINK 4.0 and earlier.

- Changed default calling convention from C-stack-based to Pascal-stack-based.

- Moved Error codes to end of file.

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 the Japanese ATM software version 2.0.3.

- Section 6, new ATM APIs has been added.

- Parts of the document have been modified to correspond to ATM software release version 3.5 and Japanese version. Section 7 contains additional error codes returned by new API functions.
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.

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 functions corresponding to ATM API version 4 have been added.
- ATM API names were updated to be consistent with the version 4 functions.
- An error code argument was added to the ATM API show functions; consequently, there were minor changes to the existing sample code.
- The conversion: \((\text{Fixed}) \ (q * 65536 + 0.5)\) in the sample code was changed to: \((\text{Fixed}) \ \text{floor} \ (q + 0.5) * 65536.0\).

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 part number changed from ATM0002 to LPS5072.
Appendix B: ATM API
Usage Table

The following table lists all available API functions for ATM 4.0 and lists usage for specific types of fonts.

- Column 1 lists the ATM API functions.
- Column 2 lists the *Font Type* which specifies whether the function is usable with single-byte (S) fonts, double-byte (D) fonts or both.
- Column 3 distinguishes a function as applicable to a *multiple master* font. These functions are defined in detail in Technical Note 5074, “Adobe Type Manager Software API With Multiple Master Fonts: Macintosh.”
- Column 4, the Notes column, specifies whether a function is CID-keyed font specific (1), will return an error for fonts with protected character outlines (2), or requires special consideration for the Macintosh PowerPC™ (3).

<table>
<thead>
<tr>
<th>API Functions</th>
<th>Font Type</th>
<th>Multiple Master Specific</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>addMacStyleToCoordsATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>convertCoordsToBlendATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>copyFitATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>createPermBlendedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>createTempBlendedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>decodeBlendedFontNameATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disposeTempBlendedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disposeTempBlendedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>encodeBlendedFontNameATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>endFillATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillClosePathATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillCurveToATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillLineToATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillMoveToATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flushCacheATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API Functions</td>
<td>Font Type</td>
<td>Multiple Master Specific</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>--------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>fontAvailableATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fontFitATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getATMInfoATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAxisBlendInfoATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getBlendedFontTypeATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getBlessedFontNameATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getFontFamilyFONDATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getFontInfoATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getFontSpecsATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getGlyphOutlineATM</td>
<td>D</td>
<td></td>
<td>1,2,3</td>
</tr>
<tr>
<td>getMasterFONDATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getNumAxesATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getNumBlessedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getNumMastersATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getOutlineATM</td>
<td>S</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>getOutlineATM2</td>
<td>D</td>
<td></td>
<td>2,3</td>
</tr>
<tr>
<td>getPSNumATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getRegularBlessedFontATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>getRunsATM</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getRuns2ATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getSubstFontInfoATM</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getTempBlendedFontFileIDATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>initNameAndVersion</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>initVersionATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isSubstFontATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMFontPickerATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>showGlyphATM</td>
<td>D</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>showTextATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>showTextATMerr</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>showTextDesignATM</td>
<td>S</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>API Functions</td>
<td>Font Type</td>
<td>Multiple Master Specific</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
<td>--------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>startFillATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyshowGlyphTextATMerr</td>
<td>D</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>xyshowTextATM</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyshowTextATMerr</td>
<td>S,D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C: Style bits positions

The following table illustrates the position values of the *style bits* from the function calls `fontAvailableATM()` and `isSubstFontATM()`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>0</td>
<td>– regular style, no constant</td>
</tr>
<tr>
<td>bold</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>italic</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>underline</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>outline</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>shadow</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>condense</td>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>extended</td>
<td>64</td>
<td>6</td>
</tr>
</tbody>
</table>
Appendix D: Obsolete ATM API Functions

The following ATM API functions are obsolete and therefore are discouraged from being used. Currently there are no replacement functions.

normToUserCoordsATM

short normToUserCoordsATM(Fixed *normalCoords, Fixed *coords)

This function is no longer supported as of ATM 3.9, June 1995.

Convert the design coordinates to a range usermin..usermax instead of 0.0..1.0.

userToNormCoordsATM

short userToNormCoordsATM(Fixed *coords, Fixed *normalCoords)

This function is no longer supported as of ATM 3.9, June 1995.

Convert the design coordinates to a range 0.0..1.0 instead of usermin..usermax.

showTextATM

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

This is replaced by the showTextATMErr() function.

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

xyshowTextATM

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

This is replaced by the xyshowTextATMErr() function.

Show length characters starting at text transformed by the specified matrix. Matrix maps one point character space to device space, relative to current pen position. Matrix’s tx and ty components are updated. Character x and y widths are specified by displacements. Returns the number of characters not shown.