Contents

Examples v

Figures vii

Adobe Type Manager Software API: Windows 9

1 Introduction to Adobe Type Manager and the Adobe Type Manager Applications Programmer Interface 9

2 Application Developer's View of New ATM 4.0 Features 11
   Auto-Activation, Font Substitution, and Multiple Master Instances 11
   Printing support with ATM 4.0 for Windows 95 13
   Font Smoothing 15
   Expanded Support of the Windows API 15
   Font Management 16

3 Overview of the ATM API 16

4 The Transformation Matrix 18
   More On The Transformation Matrix 23

5 Structures, Callback Functions, and Data Types 25

6 ATM Software API Functions 29
   API Information Functions 31
   API Selection Functions 38
   API Outline/Fill Functions 42
   API TextOut Functions 51
   API Font Management Functions 55

7 ATM Return Values and Flags 62
   Return values for non-Boolean functions 62
   Return values for the font substitution database 66
   Return values for PostScript stub and .pfm generation 66
   Additional flags for the wOptions parameter 67
   Additional flags for the options parameter 67
   Type bits for ATMEnumFonts() and ATMAddFonts() 68
   mmFlags for the ATMMMMetricsHeader structure 69
   mmAxisAttrs for ATMMMMMMetricsHeader structure 70
   Various flags supported by ATMSetFlags() 71
   Return values for *lpFromOutline in ATMFontAvailable() 72
Appendix A: Changes Since Earlier Versions  73

Appendix B: Obsolete ATM API Functions  75

Index  77
Examples

Example 1  Upright text  13
Example 2  Reversed text  13
Example 3  Rotated text  14
Example 4  Skewed text  14
Example 5  Example use of \texttt{ATMGetOutline()}  37
Example 6  Example use of ATM fill functions  41
Example 7  Example use of \texttt{ATMBBoxBaseXYShowText()}  46
# Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>The skew of the coordinate system</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Upright text</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Reversed text in x and y</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Rotated text</td>
<td>14</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Skewed text</td>
<td>15</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Curves showing Bezier control points and end points</td>
<td>36</td>
</tr>
<tr>
<td>Figure 7</td>
<td>ATM Fill functions</td>
<td>42</td>
</tr>
<tr>
<td>Figure 8</td>
<td><strong>ATMBoxBaseXYShowText()</strong></td>
<td>46</td>
</tr>
</tbody>
</table>
1 Introduction to Adobe Type Manager and the Adobe Type Manager Applications Programmer Interface

Adobe Type Manager® (ATM) software for Microsoft Windows® allows Type 1 PostScript® language font programs to be used transparently by single- and double-byte Windows application programs to rasterize outline font data strings on displays and printers. The ATM 4.01 Applications Programmer Interface (API) described in this document supports all Roman language versions of ATM for Windows 3.1 and Windows 95; and the Japanese versions of ATM 2.5J and 3.2J.

Note Both Roman and Japanese language versions of the ATM API have been released to developers, without documentation, since this document was last published and released with an API in 1993. This revision (January 1997) contains information on both the documented and undocumented releases of the ATM API, as well as information on new API calls. The ATM 4.01 API supports both single- and double-byte fonts in one API.

There are two versions of ATM 4.0 for Windows 95: the standard version and the Deluxe version. The system component of each is identical and therefore the same API is used. 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-aliased 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 new additional 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, perform font substitution. In addition, ATM 4.0 for Windows 95 supports automatic font activation and substitution of multiple master fonts. These new features are discussed in Section 2. For more complete information on the features of the ATM product, see the User’s Guide.

The ATM software intercepts GDI text drawing calls and images all requests that can be satisfied using Type 1 Roman and Japanese language fonts. Using the ATM 4.01 API, an application can perform such functions as enumerating
fonts; checking font availability; returning a font’s bounding box; returning a font path, Windows’ menu name, or PostScript name; and checking internal operating flags. The API also provides a function for overriding some of the default actions of ATM. The API can be used to access a character’s outline data; the resulting Bézier curves can then be used to reconstruct the altered character, or outline, or to fill the resulting shape using Windows GDI functions. The API has functions for showing text either based on the upper left corner of the text bounding box or the baseline of the text. Finally, the API provides functions allowing an application to perform font management operations such as adding and removing fonts on-the-fly, checking or changing the status of installed fonts, and creating multiple master font instances.

The ATM 4.01 API explained in this document comprises two separate libraries—a 16-bit and a 32-bit library. Each library communicates through the appropriate DLL shipped with ATM software. The 16-bit library maintains compatibility with any Windows 3.X application or Windows 95 16-bit application using version 3.02 or newer of ATM. The 16-bit API also supports kanji calls to ATM3.2J.

The 32-bit library is designed specifically to work on Windows 95 with newer versions of ATM. For the first time, one library supports both Roman and kanji applications. ATM version 3.02 and ATM 3.2J are 32-bit capable.

**Note** Neither the ATM software nor API described in this document are compatible with Win32s or Windows NT operating systems.

The following table summarizes the file interactions described above.

<table>
<thead>
<tr>
<th>ATM API version 4.01</th>
<th>Windows 3.x</th>
<th>Windows 95</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit application</td>
<td>16-bit or</td>
<td>32-bit DLL in ATM 4.0, 3.02,</td>
</tr>
<tr>
<td>using 16-bit atm.lib</td>
<td>32-bit DLL in ATM 3.02</td>
<td>ATM 3.2J</td>
</tr>
<tr>
<td></td>
<td>or earlier</td>
<td></td>
</tr>
<tr>
<td>32-bit application</td>
<td>Not Supported</td>
<td>32-bit DLL in ATM 4.0, 3.02,</td>
</tr>
<tr>
<td>using 32-bit atm.lib</td>
<td></td>
<td>ATM 3.2J</td>
</tr>
</tbody>
</table>

This document covers information an application programmer should understand about the ATM program and using the ATM 4.01 API. Section 2 covers the functionality added to ATM 4.0 and ATM 4.0 Deluxe. 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 ATM API functions, and their relationship to the PostScript language Current Transformation Matrix (CTM), is covered in Section 4.

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

Appendix B covers the ATM API functions that are no longer recommended for use or are obsolete.

2 Application Developer's View of New ATM 4.0 Features

ATM 4.0 Deluxe for Windows 95 introduces new features and added functionality to the Windows platform. Auto-activation of a font on a document level, font substitution with the help of a metrics database, and building multiple master instances on-the-fly are features enabled by ATM 4.0. In the Deluxe version, the user can set these features “on” or “off” through the advanced settings menu in the control panel. In the standard version of ATM 4.0 which is bundled with many application programs, there is no user control of these features. Whatever feature an application developer decides to make available is automatically done within the application without user intervention.

Access to these new features is handled through some API calls which are new or updated for API 4.01 and some Windows 95 GDI escapes or calls. Since the same DLL is shipped with ATM 4.0 and ATM 4.0 Deluxe, all of the features are available to application developers.

This section is an overview of the new features. It explains how to take advantage of ATM’s new font rendering capabilities, changes to printing support with ATM, and the expanded use of the Windows API. It also discusses new font management available in ATM 4.0 Deluxe and font smoothing which is available in both ATM 4.0 versions. Neither of these new features has requires API calls.

2.1 Auto-Activation, Font Substitution, and Multiple Master Instances

The application support of font substitution, auto-activation, and generating or creating multiple master instances on-the-fly are closely related. The key characteristic that relates these features to each other is that the created fonts are not enumerated, but are realized. Enumerated, in this context, refers to the process of returning a list of font menu names (from the developer’s point-of-view) or supplying a list of fonts that are selectable from an application’s font menu (user’s point-of-view). Realized in this context is defined as the point at which the Windows Graphics Device Interface (GDI) calls ATM at the first selection of a particular font into a Device Context (DC) using the GDI call SelectObject() or the API call ATMSelectObject().
Font substitution is the action of replacing the text of an uninstalled font within a document with a simulated font having like characteristics. The simulation of the font is done with the appropriate font metrics so that line breaks and formatting are not compromised. ATM 4.0 can approximate the look of missing fonts by using a sans serif or serif multiple master font built specifically for simulation and the metric information contained in the ATM font substitution database. These fonts are called AdobeSanMM and AdobeSerMM and do not appear within an application’s font menu. Font substitution is a new feature for the Windows platform, and has been introduced as a part of ATM 4.0.

Font activation is the operation of making a particular font available to an application. In the past, this was achieved when the ATM Control Panel called ATMAddFont(). Auto-activation is the action of identifying and using a PostScript Type 1 font within a document when that font is not currently activated (available) within ATM. When a document is opened containing Type 1 fonts that have not been activated using ATM Deluxe’s Font Manager, ATM will automatically activate the font locally (within that document only) and render the text correctly.

For multiple master fonts, functionality has increased. Previously, Windows ATM version 3.X could generate multiple master instances automatically if the base multiple master font was active. These are called on-the-fly multiple master instances. Now, ATM 4.0 for Windows 95 can auto-activate a base multiple master font and then use that font to build an on-the-fly multiple master instance. In addition, the Deluxe version of ATM can now simulate a multiple master instance using the font substitution capability.

Substitution fonts, auto-activated fonts, and the on-the-fly multiple master instances are used locally and are not enumerated system wide. ATM removes the fonts automatically after the application has deleted the font object.

To determine if ATM can properly render a font that is not currently enumerated on the system, an application should use the ATM backdoor API call, ATMFontAvailable() (see Section 6.1). ATM returns True if it can render the font in any fashion, False if it cannot. To determine how ATM will render the font the value of the *lpFromOutline output parameter has been expanded from a Boolean to an integer. A non-zero value now means that ATM will be available to render the font with the correct character metrics either from the enumerated font outline itself or from the substitution font and the metrics for it in the font substitution database.

An *lpFromOutline return value of one (1) means that ATM can render the font from the actual available Type 1 font outline. The value of one (1) is returned even if the outline must first be auto-activated. It also returns a one (1) if the requested font is an on-the-fly multiple master instance of a base
multiple master font that is already active or can be auto-activated. A value of one (1) is also returned when true **Bold** or **Italic** styles of outline fonts are requested and can be rendered using an actual outline or a multiple master instance.

An *lpFromOutline* return value of two (2) means the font is not installed, but ATM can substitute the font using the multiple master substitution font and metrics from the font substitution database. ATM will create a substitution font for any multiple master instance whose base font metrics are listed in the font substitution database.

If a user specifies a font with a style that is not available in that font family, ATM returns a value of zero (0) for the *lpFromOutline* parameter. ATM can still render the style by doing what is necessary to synthesize the style, such as emboldening (or smear bolding, which means shifting outline pixels to give the effect of bold text) or shearing (which means shifting outline pixels to give the effect of italic text or character depth). However, the metrics of these synthetic styles do not match the true font metrics. While the flag is set to zero (0), the main return code to **ATMFontAvailable()** will be True because ATM can render the style.

Older applications will continue to treat *lpFromOutline* as a Boolean.

Following is a table showing the expected output from a call to **ATMFontAvailable()**:

<table>
<thead>
<tr>
<th>Type of Font</th>
<th>Return value</th>
<th>*lpFromOutline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Fonts</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Multiple master instances on-the-fly</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Bold styles of multiple master fonts with a weight axis</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Auto-activation fonts</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Substitution fonts</td>
<td>True</td>
<td>2</td>
</tr>
<tr>
<td>Synthetic fonts</td>
<td>True</td>
<td>0</td>
</tr>
<tr>
<td>Fonts that cannot be rendered by ATM</td>
<td>False</td>
<td>0</td>
</tr>
</tbody>
</table>

### 2.2 Printing support with ATM 4.0 for Windows 95

If an application uses the **TextOut()** Windows GDI call when printing auto-activated fonts, substituted fonts, or on-the-fly multiple master instances, each will print without any application changes to support ATM 4.0. The fonts are downloaded to the printer when needed.
If an application generates its own PostScript code, it is important that it uses the \texttt{GETFACENAME} escape to determine the name of the font that will be downloaded by the PostScript driver and the \texttt{DOWNLOADFACE} escape to download the outlines.

\textbf{Note} Escapes such as \texttt{GETFACENAME} and \texttt{DOWNLOADFACE}, and others mentioned in this document, are supplied as part of the Windows GDI. More information on escapes can be found in the Microsoft Windows Software Development Kit.

ATM 4.0 (either version) now supports the mechanism originally designed to support TrueType fonts on a Windows 95 system. By parsing the \texttt{ATM.INI} and \texttt{WIN.INI} files, the PostScript driver creates a table of registered fonts. In other words, the driver consults these files in order to create its own table of all fonts that are available for downloading. This becomes the driver's registry. If a font is not registered for the driver, yet can be rendered by ATM, the PostScript driver will ask ATM for the font information and construct a font on-the-fly. This type of font always has a unique name starting with the characters \texttt{MSTT}. The constructed font is downloaded in the print job either automatically when the driver sees a \texttt{TextOut()} call or at the application's request when the driver sees a \texttt{DOWNLOADFACE} escape.

\textbf{Note} The Adobe PostScript driver (version 4.1 and above) will parse both the \texttt{ATM.INI} and the \texttt{WIN.INI} for fonts available on the system. The Microsoft PostScript driver (version 4.0 and above) parses only the \texttt{WIN.INI} file. Therefore, results filed in the driver registry will differ depending on which driver is selected at start-up.

Auto-activated fonts, substituted fonts and on-the-fly multiple master instances are not registered for either PostScript driver but are printed in this fashion. To ensure that the PostScript code uses the correct font, it is important to first determine the name of the font the PostScript driver will download and generate the PostScript code accordingly.

The location of font files can be found by calling the ATM API call, \texttt{ATMGetFontPaths()} (see Sections 3 and 6.1). This call returns the paths and file names of the Type 1 outline font files. With the font file path, the font outline can be downloaded in a PostScript data stream using the \texttt{PASSTHROUGH} escape. However, downloading fonts in this fashion does not allow the PostScript driver to manage VM. For VM management, it is more reliable to use the \texttt{DOWNLOADFACE} escape to download a PostScript driver-registered font file to the printer.

\textbf{Note} The font constructed on-the-fly by the driver is built using font information on a character-by-character basis and is not identical to the Type 1 outline font. The incremental download of the \texttt{MSTT} font is used for unregistered fonts.
In the case of multiple master instances, the font path and file name returned by `ATMGetFontPaths()` point to the base multiple master font. A PostScript-language `findfont` for a particular multiple master instance will succeed provided the base multiple master font is available in VM. When printing substituted fonts, the PostScript driver should call the Windows GDI and let it download the *MSTT* (driver created) font.

### 2.3 Font Smoothing

ATM 4.0 can *smooth* (produce anti-aliased) Type 1 fonts on screen when it is given a color device context (DC) for both standard and multiple master faces. If an application draws text off screen as a monochrome bitmap, ATM cannot smooth (anti-alias) 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. Font smoothing works best on high resolution color devices (displays that are 16-bit and higher). At 4- and 8-bit color depths, font smoothing is available only for black text and looks best on a white background.

At this time, no backdoor calls have been written to initiate font smoothing from an application. The API function `ATMSetFlags()` can be used to query the current ATM setting. Within the ATM 4.0 Deluxe user interface on the advanced settings menu the user may turn the font smoothing feature on and off on a system-wide basis. In the standard version of ATM 4.0, font smoothing is “on” by default.

### 2.4 Expanded Support of the Windows API

ATM 4.0 supports the following calls to the Windows API: `GetOutlineTextMetrics()`, `GetGlyphOutline()`, `GetTextMetrics()`, `GetCharWidth()`, `GetKerningPairs()`, and `GetCharABCWidths()`. These calls can be used with all Type 1 fonts including auto-activated and substituted fonts. It should be noted that these functions do not work when a Type 1 font is realized by the PostScript driver. Applications can control this by using the `ATMSelectObject()` backdoor call instead of making a call to the driver.

In addition, the following escapes are supported:

- `GETEXTENDEDTEXTMETRICS`
- `GETPAIRKERNTABLE`

**Note**  Windows 95 has changed the *Escape* function to *ExtEscape* for 32-bit applications. Please use the *ExtEscape* when working with Type 1 fonts.

**Note**  The Windows 95 GDI call `SetFontData()` only works with TrueType fonts.
2.5 Font Management

ATM 4.0 Deluxe for Windows 95 has a new user interface for Type 1 and TrueType font installation and management. These capabilities are not available in the standard version of ATM 4.0. Through the ATM 4.0 Deluxe user interface (control panel), fonts can be turned off and on without uninstalling or reinstalling them, fonts can be viewed on screen or printed in sample sheets, groups of fonts can be created that are called font sets, and font sets can be exported for use by other applications. Enumeration of fonts can be controlled by these font sets. The font management control allows the addition or deletion of fonts individually or by sets. No API functions are needed to take advantage of the new font management capabilities.

3 Overview of the ATM API

For the first time, one API library supports both single- and double-byte font programming needs. The ATM 4.01 API also includes both 16-bit and 32-bit libraries. Single- and double-byte support is available in each of these libraries. Another important point about this API is that there is only one DLL linking calls from it to the ATM 4.0 and ATM 4.0 Deluxe programs. Both libraries access the DLL which is included in ATM 4.0 and 4.0 Deluxe. While the user interface is different in the standard and Deluxe versions of the program, full functionality is available to the developer in either version of ATM.

The 16-bit library is available to maintain compatibility with 16-bit (Windows 3.X) applications. The 16-bit library will work on either Windows 3.X or Windows 95. For Windows 95 use, it accesses the 32-bit DLL in ATM 4.0 or ATM 4.0 Deluxe. It can also use the ATM 3.2J DLL, and the 16-bit or 32-bit DLL that shipped in ATM 3.02.

The 32-bit library is designed specifically to work on Windows 95. It uses the 32-bit DLL that comes with ATM 4.0 or ATM 4.0 Deluxe. It can also use the DLL shipping with ATM 3.2J.

Note The ATM 4.01 API 32-bit library will not work with Win32s or Windows NT.

Several API functions exist only in the 16-bit library. It is recommended that these functions not be used for 32-bit application development. In certain cases, there are now 32-bit functions that can be used to accomplish the same tasks. For other application needs, Windows GDI calls or escapes should be used. The ATM API calls, their replacements and workarounds are covered in detail in Sections 6. There are currently no plans to upgrade these 16-bit functions in the 32-bit library, though they will be maintained in the 16-bit library for application compatibility.
Note The following font file types mentioned throughout this document. A .pfb file contains a standard or multiple master Type 1 font outline. A .pfm file contains standard or multiple master instance font metrics information. A .mmm file contains font specific multiple master metrics information. The font stub file is identified by the .pss extension. The table below summarizes the font file types information.

<table>
<thead>
<tr>
<th>Font Type</th>
<th>Outline File</th>
<th>Metrics File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Master Font</td>
<td>.PFB</td>
<td>.PFM</td>
</tr>
<tr>
<td>Multiple Master Font</td>
<td>.PFB</td>
<td>.MMM</td>
</tr>
<tr>
<td>Multiple Master Instance</td>
<td>.PSS</td>
<td>.PFM</td>
</tr>
<tr>
<td>Multiple Master Default Instance</td>
<td>.PFB</td>
<td>.PFM</td>
</tr>
</tbody>
</table>

The ATM 4.01 API provides a number of new and updated functions. These are briefly introduced in this section. More complete information on all of the functions, and their associated data structures and flags is located in Sections 5, 6, and 7 of this document.

ATMFontAvailable() checks whether or not a Type 1 font outline is available and can be rendered.

ATMFinish() is a new 32-bit only function that unloads the backdoor library (removes it from virtual memory).

ATMEnumFonts() enumerates all fonts in ATM, which may include Type 1, multiple master Type 1, and multiple master instances.

ATMEnumMMFonts() enumerates all multiple master base fonts and font instances.

ATMGetFontPaths() takes a font name and font style, then returns the associated font path and file name.

ATMGetMenuName() takes a PostScript font name and returns the equivalent Windows font name and style.

ATMGetPostScriptName() takes a Windows font name and a font style, then returns the equivalent PostScript name for the font.

ATMGetBuildStr() returns the ATM build string. The returned string includes encoded information such as the version number, the DLL type, the build number, and the release name.

ATMSelectObject() is designed to replace the older functions ATMClearAllForce(), ATMForceCreate(), ATMForceDefer(), and ATMForceExactWidth(). It allows applications to override some of ATM’s
default actions as specified by the user through the ATM Control Panel. This is handled through a series of options flags given to ATMSelectObject() that perform similar actions to the older functions. These flags are described in Section 7.5.

**ATMForceFontChange()** forces a WM_FONTCHANGE message to be sent when ATMEndFontChange() is called regardless of whether or not a font was added (using ATMAddFont()).

**ATMMakePFM()** constructs a .pfm (metrics) file with blended metrics. This requires a font name and font style for the requested multiple master font instance to be constructed.

**ATMMakePSS()** can construct a .pss (stub) file for the PostScript driver. This also requires a font name and font style for the requested multiple master font instance to be constructed. This function is used only for multiple master fonts.

**ATMSetFlags()** is used to examine some of ATM’s internal operating flags such as ATM_DOWNLOAD, ATM_USEDEVFONTS, ATM_SUBSTITUTE, ATM_ANTIALIAS, ATM_AUTOACTIVATE, and ATM_GDIFONTS.

The functionality of **ATMGetOutline()** has been extended to allow an application to obtain the PostScript language hinted outline description of both Type 1 Roman fonts and Japanese language fonts. **ATMGetOutline()** replaces the older, Japanese-only function **ATMGetOutline2()**.

Rendering and filling are not available in the 32-bit library. Windows GDI calls should be used to replace the 16-bit functions ATMFillStart(), ATMFillMoveTo(), ATMFillLineTo(), ATMFillCurveTo(), ATMFillClosePath(), ATMFillGetBBox(), ATMFillEnd(), and ATMBaseFillEnd().

For more complete information on all of the functions now supported in the ATM 4.01 API, see Section 6 of this document.

## 4 The Transformation Matrix

The ATM software allows applications to specify the work of transforming (translating, scaling, rotating, and skewing) text. The application specifies the transformation by providing a matrix of type ATMFixedMatrix() to the ATMXYShowText( ), ATMBBoxBaseXYShowText(), and ATMGetOutline() functions. This matrix maps the coordinate system onto the coordinate system of the device context currently in use.
The matrix is similar (but not identical) to the matrix passed to the PostScript language operator \texttt{makefont}. The values used in this transformation matrix are \textit{device} units (pixels). The differences between this transformation matrix and the matrix normally passed to the \texttt{makefont} operator are discussed below in Section 4.1.

Note that \texttt{ATMFixedMatrix()} contains variables \textit{tx} and \textit{ty}, which are not currently used by ATM software but are included for completeness. Currently, \textit{tx} and \textit{ty} should be initialized to \textit{zero} (0).

The matrix values are as follows:

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

The elements \textit{a} and \textit{d} are the horizontal and vertical scaling factors. \textit{Mx} is the font size in the \textit{x} direction. For example, if \textit{Mx} is 12 then the \textit{x} dimension of the font will be that of a 12 pixel font. \textit{My} is the font size in the \textit{y} direction. The \textit{y} dimension of the font will be that of an \textit{My} pixel font. The elements \textit{b} and \textit{c} determine the orientation of the axes relative to a normal Cartesian coordinate system. Element \textit{b} includes the skew of the axis in the \textit{x} direction.

\textbf{Figure 1} illustrates this geometry. If \textit{\alpha} is the upward skew of the transformed \textit{x} axis relative to the normal \textit{x} axis, then \( b = -M_x \tan (\alpha) \). Similarly if \textit{\beta} is the clockwise vertical skew of the transformed \textit{y} axis relative to the normal \textit{y} axis, then \( c = M_y \tan (\beta) \). Note that all these values have been adjusted to map into a coordinate system in which \textit{y} values increase downward and the \textit{x} and \textit{y} units have the same physical dimensions.

\textbf{Figure 1} \textit{The skew of the coordinate system}
In Figure 1, the skew of the coordinate system is specified by the parameters \( \alpha \) and \( \beta \). The directions of skew indicated in the drawing would result in positive values of both \( \alpha \) and \( \beta \).

A slightly different formula is easier for uniform rotations (using polar coordinates). If a size of \( M \) pixels with a counterclockwise rotation of \( \theta \) degrees is desired, then

\[
\begin{align*}
a &= M \cos \theta \\
b &= -M \sin \theta \\
c &= -M \sin \theta \\
d &= -M \cos \theta
\end{align*}
\]

**Examples**

The following pages contain some simple code examples showing how the matrix should look when passed into `ATMXYShowText()`.
Example 1: *Upright text*

\[
\begin{align*}
m.a &= \text{ATMINTTOFIXED}(126); \quad /* \text{Upright 126 pixel font} */ \\
m.b &= 0; \\
m.c &= 0; \\
m.d &= \text{ATMINTTOFIXED}(-126); \\
m.tx &= 0; \\
m.ty &= 0;
\end{align*}
\]

**Figure 2** *Upright text*

![Upright text](image)

Example 2: *Reversed text*

\[
\begin{align*}
m.a &= \text{ATMINTTOFIXED}(-55); \quad /* \text{Reversed in x. 55 pixels in x.} */ \\
m.b &= 0; \\
m.c &= 0; \\
m.d &= \text{ATMINTTOFIXED}(85); \quad /* \text{Reversed in y. 85 pixels in y.} */ \\
m.tx &= 0; \\
m.ty &= 0;
\end{align*}
\]

**Figure 3** *Reversed text in X and Y*

![Reversed text in X and Y](image)
Example 3: Rotated text

```c
#define DEGTORAD (2 * 3.14159/360)
/* conversion from degrees to radians */
#define FIX(q) (ATMFixed) (q * 65536.0)

m.a = FIX (44 * cos(25 * DEGTORAD)); /* 44 pixel font */
m.b = FIX (-44 * sin(25 * DEGTORAD)); /* rotated 25 degrees counter-clockwise */
m.c = FIX (-44 * sin(25 * DEGTORAD));
m.d = FIX (-44 * cos(25 * DEGTORAD));
m.tx = 0;
m.ty = 0;
```

Figure 4 Rotated text

Example 4: Skewed text

```c
#define DEGTORAD (2 * 3.14159/360)
/* conversion from degrees to radians */
#define FIX(q) (ATMFixed) (q * 65536.0)

m.a = FIX (44);
/* 44 pixels in x*/
m.b = FIX (-44 * tan(15 * DEGTORAD));

/* X axis 15 degrees counter-clockwise */
m.c = FIX (73 * tan(35 * DEGTORAD));

/* Y axis 35 deg clockwise*/
m.d = FIX (-73);
/* 73 pixels in y*/
m.tx = 0;
m.ty = 0;
```
4.1 More On The Transformation Matrix

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 $\frac{1}{72}$ 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 (1) 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 $^\prime$ (CTM prime) which corresponds to the mapping of the new coordinate system into device space. This new CTM $^\prime$ is calculated as follows:

$$CTM^\prime = T \times CTM$$

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 a coordinate system which has a screen pixel of $\frac{1}{72}$ inch. In this case, there is no scaling necessary to map user space into device space. Additionally, assume that user space has a coordinate system in which $y$ increases going down (as in the Windows video coordinate system), while the PostScript
language coordinate system has \( y \) increasing up the screen. In order 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:

\[
T = \begin{bmatrix}
M_x & (M_x \tan(\alpha)) & (M_y \tan(\beta)) & M_y & 0 & 0
\end{bmatrix}
\]

where \( M_x \) is the scale factor in \( x \), \( M_y \) is the scale factor in \( y \) and \( \alpha \) and \( \beta \) are the skew of the transformed \( x \) and \( y \) axes respectively. \textbf{Figure 1} indicates this geometry.

When we concatenate \( T \) and the CTM we get

\[
\text{CTM}' = \begin{bmatrix}
M_x & (-M_x \tan(\alpha)) & (M_y \tan(\beta)) & -M_y & 0 & 0
\end{bmatrix}
\]

This is identical to the matrix specified above. This is the matrix passed to the \texttt{ATMXYShowText()}, \texttt{ATMBBoxBaseXYShowText()}, and \texttt{ATMGetOutline()} calls.

Note that the size of the font is built into \( M_x \) and \( M_y \) and is specified in pixels. If a font is to be displayed on screen in logical inches, and the user interface is specified in points (1/72 inch), then an application program can set \( M_x \) and \( M_y \) by scaling the user input for the \( x \) point size by \( \text{LOGPIXELSX} / 72 \) and by scaling input for the \( y \) point size by \( \text{LOGPIXELSY} / 72 \).

To calculate the matrix for rotated text, note that for \textit{uniform} rotation, the angle \( \beta = -\alpha \). If we write that \( M_x = M \cos(\alpha) \) and \( M_y = M \cos(\beta) \) (\( M \) is the point size of the font we wish to rotate), then the CTM can be written as:

\[
\text{CTM}' = \begin{bmatrix}
M \cos(\alpha) & -M \sin(\alpha) & -M \sin(\alpha) & -M \cos(\alpha) & 0 & 0
\end{bmatrix}
\]

just as indicated in the equation shown earlier in this document.

For more information on transformations of coordinate spaces, see the \textit{PostScript Language Reference Manual, Second Edition}. 

24 Adobe Type Manager Software API: Windows (24 January 1997)
5 Structures, Callback Functions, and Data Types

The following structures, callback functions, and data types are used by the functions defined in the ATM 4.01 API library. They are supported in both the 16-bit and 32-bit libraries and for single- and double-byte fonts.

ATMFixed typedef long int ATMFixed;

ATMINTTOFIXED #define ATMINTTOFIXED(n) ((ATMFixed) (((long int)(n))<< 16))

The ATM software uses the ATMFixed data type internally. ATMFixed is a 32-bit signed quantity with the fractional part of the value contained in the low order word and the integer part of the value contained in the high order word. To convert a floating point value to ATMFixed, multiply the floating point value by 65536.0 then cast the temporary result to (ATMFixed).

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

ATMFixedPoint typedef struct {
    ATMFixed x, y;
} ATMFixedPoint, ATMFAR *LPATMFixedPoint;

ATMShortPoint typedef struct {
    short int x, y;
} ATMShortPoint, ATMFAR *LPATMShortPoint;

ATMBBox typedef struct {
    ATMShortPoint ll, ur;
} ATMBBox, FAR *LPATMBBox;

The ATMBBox field names ll (lower left) and ur (upper right) can be confusing. For example, when used with ATMGetFontBBox(), if the mapping mode is MM_TEXT and the bounding box goes from (0,0) to (100,100), then the values in ll will be the point (0,0) and the values in ur will be the point (100,100). On a display in MM_TEXT mapping mode, (0,0) is actually the upper left not lower left.
ATMFontSpec  typedef struct {
    char    faceName [LF_FACESIZE];
    WORD    styles;
} ATMFontSpec, ATMFAR *LPATMFontSpec;

ATMEnumFontProc  typedef BOOL
    (ATMCALLBACK ATMEnumFontProc) (
        LLOGFONT lpLogFont,
        LPSTR lpPostScriptName,
        WORD   flags,
        DWORD  dwUserData);

typedef ATMEnumFontProc ATMFAR *LPATMEnumFontProc;

ATMMMMetricsHeader  typedef struct {
    WORD    mmVersion;
    WORD    mmFlags;
    char    mmCopyright[72];
    char    mmFontVersion[8];
    char    mmPSFontName[80];
    WORD    mmAxisAttrs;
    BYTE    mmFamily;
    BYTE    mmEncoding;
    WORD    mmCharSet;
    WORD    mmFirstChar;
    WORD    mmLastChar;
    WORD    mmDefaultChar;
    WORD    mmBreakChar;
    WORD    mmNumKernPairs;
    WORD    mmNumKernTracks;
    WORD    mmNumUnencodedChars;
    WORD    mmSizeUnencodedCharNames;
    WORD    mmNumUnencodedKernPairs;
    short   mmMasterUnits;
    short   mmMasterHeight;
    DWORD   mmSizeDesignPoint;
    WORD    mmNumAxes;
    WORD    mmNumBaseDesigns;
    WORD    mmNumPrimaryDesigns;
    char    mmFileNamePrefix[6];
    char    mmMenuName[MI_FACESIZE];
    DWORD   mmSizeDVSubrs;
    DWORD   mmReserved[19];
} ATMMMMetricsHeader, ATMFAR *LPATMMMMetricsHeader;
**ATMEnumMMInstInfo**

typedef struct {
    LPSTR lpszInstName;
    WORD styles;
} ATMEnumMMInstInfo, ATMFAR *LPATMEnumMMInstInfo;

**ATMEnumMMFontProc**

typedef BOOL (ATMCALLBACK ATMEnumMMFontProc) (
    LPVOID lpEnumData,
    DWORD dwUserData);

typedef ATMEnumMMFontProc ATMFAR *LPATMEnumMMFontProc;

**ATMFontPathInfo**

typedef struct {
    char pfbPath[ATM_MAXPATHLEN];
    char pfmPath[ATM_MAXPATHLEN];
    char mmmPath[ATM_MAXPATHLEN];
    char pssPath[ATM_MAXPATHLEN];
} ATMFontPathInfo, ATMFAR *LPATMFontPathInfo;

**ATMClosePathProc**

typedef BOOL (ATMCALLBACK ATMClosePathProc) (
    DWORD dwUserData);

typedef ATMClosePathProc *LPATMClosePathProc;

This callback function is used with the **ATMGetOutline()** function. See Section 6.3 for more information.

**ATMMoveToProc**

typedef BOOL (ATMCALLBACK ATMMoveToProc) (
    LPATMFixedPoint lpFixPnt,
    DWORD dwUserData);

typedef ATMMoveToProc *LPATMMoveToProc;

This callback function is used with the **ATMGetOutline()** function. See Section 6.3 for more information.
**ATMLineToProc**

```c
typedef BOOL
  (ATMCALLBACK ATMLineToProc)(
   LPATMFixedPoint lpFixPnt,
   DWORD   dwUserData);
```

typedef ATMLineToProc *LPATMLineToProc;

This callback function is used with the **ATMGetOutline()** function. See Section 6.3 for more information.

**ATMCurveToProc**

```c
typedef BOOL
  (ATMCALLBACK ATMCurveToProc)(
   LPATMFixedPoint lpFixPnt1,
   LPATMFixedPoint lpFixPnt2,
   LPATMFixedPoint lpFixPnt3,
   DWORD   dwUserData);
```

typedef ATMCurveToProc *LPATMCurveToProc;

This callback function is used with the **ATMGetOutline()** function. See Section 6.3 for more information.
6 ATM Software API Functions

The ATM 4.01 API functions are presented in five sections.

- Section 6.1, API Information Functions, comprises the following functions:
  
  ATMProperlyLoaded() ATMGetFontPaths()
  ATMFinish() ATMGetMenuName()
  ATMEnumFonts() ATMGetPostScriptName()
  ATMEnumMMFonts() ATMGetVersion()
  ATMFontAvailable() ATMGetBuildStr()
  ATMFontSelected() ATMSetFlags()
  ATMGetFontBBox()

- Section 6.2, API Selection Functions, comprises the following functions:
  
  ATMSelectObject() ATMForceCreate()
  ATMClearAllForce() ATMForceDefer()
  ATMForceExactWidth()

Section 6.3, API Outline/Fill Functions, comprises the following functions:

  ATMGetOutline() ATMFillClosePath()
  ATMFillStart() ATMFillGetBBox()
  ATMFillMoveTo() ATMFillEnd()
  ATMFillLineTo() ATMBaseFillEnd()
  ATMFillCurveTo()

- Section 6.4, API TextOut Functions, comprises the following functions:
  
  ATMXYShowText() ATMBBoxBaseXYShowText()
• Section 6.5, API Font Management Functions, comprises the following functions:

ATMBeginFontChange() ATMForceFontChange()
ATMFontStatus() ATMEndFontChange()
ATMAddFont() ATMMakePFM()
ATMRemoveFont() ATMMakePSS()

Note  Unless otherwise noted, API functions that return integer values - as in extern int ATMAPI ATMxxx() - use ATM_NOERR to indicate success. In addition to the specific values noted with each function, the following values can also be returned: ATM_ERROR, ATM_CANTHAPPEN, ATM_NOTSUPPORTED, ATM_NOTRUNNING, ATM_BADPARM, and ATM_BADSTYLE.

Note  The functions listed in Section 6 are contain a version number. This number indicates the earliest version of ATM software that implements the function. All subsequent versions of ATM software will also support the function, except where noted.
6.1 **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* All API functions in this section work the same for both the 16-bit and 32-bit libraries, except where noted.

**ATMProperlyLoaded**

`extern BOOL ATM API ATMProperlyLoaded (void);`

*Note: Version 1.0*

`ATMProperlyLoaded()` initializes the backdoor and verifies whether or not ATM is installed and running.

The return value for `ATMProperlyLoaded()` is `True` if ATM is installed and running. Otherwise, the return value is `False`.

*Note* If calls are made to any ATM API functions before a call to `ATMProperlyLoaded()` is made, they will fail. In addition, the return value must be `True` in order for the subsequent ATM API functions to work properly.

**ATMFinish**

`extern int ATM API ATMFinish (void);`

*Note: Version 3.02*

This API function exists only in the 32-bit library.

`ATMFinish()` unloads the backdoor.

Failure return values are as follows:

- `ATM_NOTRUNNING` if the ATM library is not initialized.
- `ATM_BUSY` if a callback has not returned.

*Note* `ATMFinish()` must be called when the host program exits or there could be a memory leak.
**ATMEnumFonts**

```c
extern int ATMAPI ATMEnumFonts(
    LPATMEnumFontProc lpProc,
    DWORD dwUserData);
```

**Note**: Version 2.6

This routine enumerates all of ATM’s fonts. These fonts may be Type 1 fonts, multiple master Type 1 fonts, or multiple master instances.

The callback function, `ATMEnumFontProc()`, pointed to by `lpProc`, is passed once for each font. The callback is passed with a LOGFONT for the font, a pointer to the PostScript name of the font, and a flag that indicates the type of font.

The currently supported font types are `ATM_TYPE1`, `ATM_MMTYPE1`, and `ATM_MMINSTANCE`.

Return values are as follows:

- **ATM_CANCELLED** if the callback returns zero (0), in which case the enumeration halts.
- **ATM_BUSY** if another callback has not returned (32-bit only).

**Note**: If no fonts are installed, the callback will not be passed, and this function will return **ATM_NOFONTS**.

**ATMEnumMMFonts**

```c
extern int ATMAPI ATMEnumMMFonts(
    LPSTR lpszMMMenuName,
    LPATMEnumMMFontProc lpProc,
    DWORD dwUserData);
```

**Note**: Version 2.6

This function enumerates installed multiple master base fonts and instances. To enumerate base fonts, the caller passes in a NULL menu name, and a callback function, `ATMEnumMMFontProc()`, receives a pointer to an `ATMMMMetricsHeader` structure (see Section 5). To enumerate instances, the caller passes in a multiple master base font menu name, and the callback function receives the pointer `lpProc` to an `ATMEnumMMInstInfo` structure (see Section 5).

The return values are as follows:

- **ATM_BUSY** if another callback has not returned (32-bit only).
ATMFontAvailable

extern BOOL ATMAPI ATMFontAvailable ( 
    LPSTR lpFacename, 
    int nWeight, 
    BYTE cItalic, 
    BYTE cUnderline, 
    BYTE cStrikeOut, 
    int ATMFAR*lpFromOutline); 

Note: Version 1.0

ATMFontAvailable() checks whether a Type 1 font outline is available and can be rendered. All of the parameters except *lpFromOutline correspond to parameters passed to the Windows GDI function CreateFont().

The lpFacename parameter is a long pointer to the face name of the font to be verified. ATMFontAvailable() returns False if ATM can not respond to a request for the given font. If the function returns True, then ATM can respond in some way to the given font. See the description of the *lpFromOutline parameter, below.

The nWeight parameter is the weight of the font whose availability is being queried. nWeight corresponds to the same nWeight value passed to the Windows GDI CreateFont() function. Creating a logical font with a weight value of 0 to 500 is recognized as a regular weight font by the ATM software. Weight values of 501 and higher are considered bold weights by the ATM software.

The cItalic flag should be set to one (1) if the query is for an italic outline. Otherwise, the flag should be set to zero (0).

The cUnderline flag is not currently in use but is included for completeness and possible future use. To assure future compatibility, cUnderline should be set to one (1) if the desired font is to be underlined. Otherwise, cUnderline should be set to zero(0).

The cStrikeOut flag is not currently in use but is included for completeness and possible future use. To assure future compatibility, cStrikeOut should be set to one(1) if the desired font is to be struck out. Otherwise, cStrikeOut should be set to zero(0).

The semantics of *lpFromOutline have been extended in the ATM 4.01 API. *lpFromOutline is now non-zero if ATM has correct metrics for the style. When ATMFontAvailable() is True, the value of *lpFromOutline is one of the following:

ATM_SYNTH (0) means that ATM can render a font in the same family. The style may be synthesized by smearing and/or skewing. The metrics are not exact.
ATM_REAL (1) means that ATM can render the requested font and style from a true outline. The metrics are correct.

ATM_SUBST (2) means that ATM can create a substitute font for the requested font and style. The metrics are correct.

**Note** Be aware that ATMFontAvailable() does not parse the font to assure that it is a valid Type 1 font definition. Note that in the current version of the ATM software, ATMFontAvailable() does not look at the face names in the Aliases section of the ATM.INI file. Therefore, although ATM software can image a font when an aliased name such as Tms Rmn is requested, ATMFontAvailable() will return False if information about the aliased face name is requested. See the Adobe Type Manager Technical Reference Guide for more information on the ATM.INI file.

ATMFontSelected extern BOOL ATMAPI ATMFontSelected(
    HDC hDC);

**Note: Version 1.0**

ATMFontSelected() can be used to determine whether or not the currently selected font is an ATM font.

The hDC parameter specifies a handle to the device context to be queried.

ATMFontSelected() returns True if the currently selected font in the hDC is an ATM font. Otherwise, it returns False. For example, a Type 1 font realized by a PostScript driver will cause ATMFontSelected() to return False.

ATMGetFontBBox extern int ATMAPI ATMGetFontBBox(
    HDC hDC,
    LPATMBBox lpFontBBox);

**Note: Version 2.01**

This routine obtains the font bounding box for the font currently selected into the hDC.

The lpFontBBox parameter is expressed in unscaled character units.

ATMGetFontBBox returns ATM_BADPARAM if lpFontBBox is NULL.
ATMGetFontPaths extern int ATMAPI ATMGetFontPaths(
    LPATMFontSpec    lpFontSpec,
    LPATMFontPathInfo lpFontPaths);

Note: Version 2.6

Given a font name and style via the lpFontSpec parameter,
ATMGetFontPaths() returns the font path and file names in the structure
pointed to by the lpFontPaths parameter.

The following list indicates which paths will be nonempty for particular types
of fonts:

For ATM_TYPE1, the PFB (outline file) and PFM (metrics file) paths will
be non-empty.

For ATM_MMTYPE1, the PFB and MMM (multiple master metrics file)
paths will be non-empty.

For ATM_MMINSTANCE, the PFB and MMM paths will be non-empty.

Note pssPath is not currently used.

Failure return values (see Sections 7.1 and 7.2) for ATMGetFontPaths() are
as follows:

ATM_NOFONT
ATM_NOFONTFILE

ATMGetMenuName extern int ATMAPI ATMGetMenuName(
    LPSTR      lpPostScriptName,
    LPATMFontSpec lpFont);

Note: Version 2.6

ATMGetMenuName() takes a PostScript name and looks up the Windows
equivalent name. This function looks at the list of currently installed active
and inactive fonts and, potentially, at the font substitution database.

lpPostScriptName is the input parameter, which is a pointer to a string
containing the PostScript name of the font.

The Windows name and style bits are returned in the LPFont parameter. The
ATM_ITALIC and ATM_BOLD bits indicate the style.
Possible return values (see Sections 7.1 and 7.2) for `ATMGetMenuName()` include the following:

- `ATM_NOTFOUND`
- `ATM_NODATABASE`
- `ATM_DBERROR`
- `ATM_BADPSNAME`

### ATMGetPostScriptName

`extern int ATMIAPI ATMGetPostScriptName (LPATMFontSpec lpFontSpec, LPSTR lpPostScriptName);`

*Note: Version 2.6*

Given a font name and a style via the `lpFontSpec` parameter, this function returns the PostScript name in the buffer pointed to by the `lpPostScriptName` parameter. This buffer must be `ATM_PSNAMESIZE` bytes in size (see Section 7.6). This function looks at the list of currently installed active and inactive fonts and, potentially, at the font substitution database.

Possible return values for this function include `ATM_NOTFOUND` (see Section 7.1).

### ATMGetVersion

`extern WORD ATMIAPI ATMGetVersion (void);`

*Note: Version 1.0*

`ATMGetVersion()` returns the currently running ATM software version number. This is useful if an application requires a specific version of the ATM software.

It returns the major and the minor version numbers. The major version number is in the low order byte and the minor version number is in the high order byte, similar to the format of the Windows GDI call `GetVersion()`.
ATMGetBuildStr  extern void ATMAPI ATMGetBuildStr ( 
    LPSTR lpBldStr);

>Note: Version 4.00

This function returns the ATM build string in the \texttt{lpBldStr} parameter. The build string is intended for display to users, and includes encoded information on version number, DLL type, build number, release name, among other items.

For example, the string \texttt{“R v4.00-32S058G06NN”} is used for ATM 4.00.

ATMSetFlags  extern WORD ATMAPI ATMSetFlags( 
    WORD flags, 
    WORD mask);

>Note: Version 2.6

\texttt{ATMSetFlags()} is used to modify or examine some of ATM’s internal operating flags (see Section 7.9).

The \texttt{mask} parameter specifies which flags are to be modified.

The \texttt{flags} parameter specifies the new values (on or off) for the flags specified by the \texttt{mask} parameter.

The return value indicates the former state of the specified flags. Beginning with the ATM 4.01 API, a \texttt{mask} value of zero (0) returns the value of all flags (without changing any of them).

\textbf{Note} These flags modify the global setting of ATM. The corresponding settings in ATM.INI determine the settings after reboot. This call is best used to examine current settings.

The flags that can be sent by the \texttt{flags} parameter are as follows:

\begin{verbatim}
ATM_DOWNLOAD
ATM_USEDEVFONTS
ATM_SUBSTITUTE
ATM_ANTIALIAS
ATM_AUTOACTIVATE
ATM_GDIFONTS
\end{verbatim}
6.2 API Selection Functions

The functions described in this section, except for `ATMSelectObject()`, are all 16-bit functions whose use is discouraged in the ATM 4.01 API. Instead, it is strongly recommended that only `ATMSelectObject()` be used. It is supported in both the 16-bit and 32-bit libraries.

ATMSelectObject extern int ATMAPI ATMSelectObject (HDC hDC, HANDLE hNewObject, WORD options, LPHANDLE lphOldObject);

Note: Version 2.01

Note This function is designed to replace the following functions: `ATMClearAllForce()`, `ATMForceCreate()`, `ATMForceDefer()`, and `ATMForceExactWidth()`. Instead of calling one of these functions followed by a call to `SelectObject()`, an application should call `ATMSelectObject()` with appropriate flags set in the options parameter. For this parameter, `ATM_DEFER`, `ATM_SELECT`, and `ATM_USEEXACTWIDTH` (see Section 7.5) specify the equivalent actions to `ATMForceDefer()`, `ATMForceCreate()`, and `ATMForceExactWidth()`, respectively. It is not necessary to call `ATMClearAllForce()` following a call to `ATMSelectObject()`.

`ATMSelectObject()` allows applications to override some of ATM’s default actions, including font deferral and width data adjustment for the screen, as specified by the user in the ATM Control Panel.

The handle being selected into the DC is passed in `hNewObject`.

The handle to the old object is returned in `lphOldObject`. If this parameter is NULL, the old object is not returned.

The options flags select the function that `ATMSelectObject()` performs. Its values may be `ATM_DEFER`, `ATM_SELECT`, or `ATM_USEEXACTWIDTH`.

`ATM_USEEXACTWIDTH` creates a font without adjusting the width data for the screen. This has an effect only when a font is being created for a display device context.

In order to correct for the text justification behavior of some Windows applications, ATM renders screen text compressed by about 3% along the x axis. This may be undesirable for some of the more advanced text applications. An application may avoid this compression by using the `ATM_USEEXACTWIDTH` flag.
This flag is to be used in conjunction with the Windows GDI text output functions such as `TextOut()` and `ExtTextOut()`.

`ATM_SELECT` causes the font to actually be created from an outline font even when normally deferring to a device bitmap font would be possible.

When the “Use Pre-built or Resident Bitmap Fonts” checkbox in the ATM Control Panel is checked, ATM will defer to a device bitmap font if a font creation request can be satisfied with one. If an application desires a Type 1 font regardless of the status of this box, it should use the `ATM_SELECT` flag.

`ATM_SELECT` is the complement of `ATM_DEFER` and therefore they are mutually exclusive.

If the `ATM_DEFER` flag is used, regardless of the setting of the “Use Pre-built or Resident Bitmap Fonts” checkbox in the ATM Control Panel, ATM will defer to the device font. If the device font is not available, the application will get whatever the device substitutes for the requested font.

Successful return values for `ATMSelectObject()` are as follows (see Section 7.1):

- `ATM_NEWFONTSELECTED` means that `hNewObject` is selected for the first time.
- `ATM_OLDFONTSELECTED` means that `hNewObject` had previously been selected into the same `DC`. Options will not have any effect in this case.
- `ATM_FOREIGNFONTSELECTED` means that `hNewObject` represents a non-ATM font.

Versions 2.01 and 2.02 of ATM had another successful return value for this function called `ATM_NONFONTSELECTED`. This value is no longer supported.

Failure return values for `ATMSelectObject()` are as follows (see Section 7.1):

- `ATM_NOTRUNNING`
- `ATM_BADPARM`
- `ATM_SELECTERROR`
**ATMClearAllForce**

extern BOOL ATMAPI ATMClearAllForce (void);

*Note: Version 2.0*

This function undoes preceding calls to any and all of `ATMForceCreate()`, `ATMForceDefer()`, or `ATMForceExactWidth()`, whose effects are pending. This function returns True if the call is successful; False otherwise.

*Note* Use of this function is discouraged. Use `ATMSelectObject()` instead. This function exists only in the 16-bit library.

**ATMForceCreate**

extern BOOL ATMAPI ATMForceCreate (void);

#define ATMForceSelect ATMForceCreate

*Note: Version 1.1*

This function causes the “next” call for creating a font to actually create an ATM font even when normally deferring to a device font would be possible. It is important that an application making this call not give up its time slice between calling this routine and actually creating (selecting) a font. Doing so may render the call to this routine void as another application may attempt to create a font which will be affected by the first application’s call to `ATMForceCreate()`. This function returns True if the call is successful; False otherwise.

*Note* `ATMForceCreate()` is not a toggle. The effect can be realized only once. `ATMForceCreate()` and `ATMForceDefer()` are mutually exclusive. So a call to one voids a previous call to the other. A font is actually created when it is selected into a target DC and not when `CreateFont()` or `CreateFontIndirect()` is called. Therefore, an application must call `ATMForceCreate()` immediately before calling `SelectObject()` in order to achieve the desired effects. If a given font is selected into more than one DC, then `ATMForceCreate()` must be called before each call to `SelectObject()`. Actually, it is more appropriate to use the macro `ATMForceSelect()` instead of directly calling `ATMForceCreate()`.

*Note* Use of this function is discouraged. Use `ATMSelectObject()` instead. This function exists only in the 16-bit library.
ATMForceDefer extern BOOL ATMAPI ATMForceDefer (void);

*Note: Version 2.0*

This function causes the “next” call for creating a font to necessarily be deferred to the device even when ATM would normally create the font. It is important that an application making this call not give up its time slice between calling this routine and actually creating (selecting) the font. Doing so may render the call to this routine void as another application may attempt to create a font which will be affected by the first application’s call to `ATMForceDefer()`. This function returns True if the call is successful; False otherwise.

*Note* `ATMForceDefer()` is not a toggle. The effect can only be realized once. `ATMForceCreate()` and `ATMForceDefer()` are mutually exclusive. So a call to one voids the previous call to the other. A font is actually created when it is selected into a target DC and not when `CreateFont()` or `CreateFontIndirect()` is called. Therefore, an application must call `ATMForceDefer()` immediately before calling `SelectObject()` in order to achieve the desired effects. If a given font is selected into more than one DC, then `ATMForceDefer()` must be called before each call to `SelectObject()`.

ATMForceExactWidth extern BOOL ATMAPI ATMForceExactWidth (void);

*Note: Version 1.1*

This function causes the “next” call for creating a font to create a font without adjusting the width data for the screen. The effect can only be realized for a font that is selected into a display DC. It is important that an application making this call not give up its time slice between calling this function and actually creating (selecting) the font. Doing so may render the call to this function void as another application may attempt to create a font which will be affected by the first application’s call to `ATMForceExactWidth()`. This function returns True if the call is successful; False otherwise.

*Note* `ATMForceExactWidth()` is not a toggle. The effect can only be realized once. A font is actually created when it is selected into a target DC and not when `CreateFont()` or `CreateFontIndirect()` is called. So, an application must call `ATMForceExactWidth()` immediately before calling `SelectObject()` in order to achieve the desired effects. If a given font is selected into more than one screen DC, then `ATMForceExactWidth()` must be called before each call to `SelectObject()`.

*Note* Use of this function is discouraged. Use `ATMSelectObject()` instead. This function exists only in the 16-bit library.
6.3 API Outline/Fill Functions

Using `ATMGetOutline()` and its associated callback 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. `ATMGetOutline()` works similarly to the PostScript language operator `pathforall`.

The remaining functions described in this section are all 16-bit fill functions whose use is discouraged in the ATM 4.01 API.

```c
extern int A TMPI API A TMGetOutline ( HDC hDC, WORD c, LPATMF ixedMatrix lpMatrix, L PATM MoveToProc lpProcMoveTo, L PATML ineToProc lpProcLineTo, L PATMC urveToProc lpProcCurveTo, L PATMClosePathProc lpProcClosePath, DWORD dwUserData);
#define ATMGetOutline2 ATMGetOutline
```

*Note: Version 1.0*

`ATMGetOutline()` reports the outline of the specified character. This function has been updated to replace `ATMGetOutline2()`. By changing the definition of the parameter `c` from *char* to *WORD*, `ATMGetOutline()` can now handle double-byte characters.

A PostScript language Type 1 font character can be represented by a series of the four basic elements of a path, which are the PostScript language operators *moveto*, *lineto*, *curveto*, and *closepath*. An application program supplies `ATMGetOutline()` with a callback function for each of the four elements.

Each time `ATMGetOutline()` processes one of these elements, it will execute the corresponding application-supplied callback function, with `dwUserData` and the data points that correspond to the element being processed. The application-supplied callback function can then manipulate or store the data. More specifically, the application-supplied callback can handle the filling of the constructed path, or simply assemble it for later use in the application.

The `hDC` parameter is the handle to the *DC* that identifies the font to be used.

The `c` parameter is the actual character - single-byte or double-byte - whose outline is to be retrieved.
The encoding scheme used for double-byte characters is *Shift-JIS* encoding. The character set used depends upon the PC platform on which application software is developed. ATM software Japanese version supports three OEM-dependent character sets: *78JIS NEC Extended*, *83JIS Fujitsu Additional*, and *83JIS*. These character sets are used on NEC PCs or NEC PC-compatibles, Fujitsu PCs, and IBM PC and IBM PC-compatibles, respectively. The character code is interpreted the same way the Windows GDI function `TextOut()` would interpret characters in the character array passed to it, as the properties of the currently selected font are taken into account.

The `lpMatrix` parameter is the transformation matrix that the application wants ATM software to apply to the character before the data is returned. This matrix is of the same form as the matrix used with `ATMXYShowText()` (see Sections 4 and 6.4).

The `dwUserData` parameter is a data value that the application program can use as a message. This parameter is passed by `ATMGetOutline()` to the application-supplied callback functions when they are executed. `dwUserData` can be NULL.

`lpProcMoveTo`, `lpProcLineTo`, `lpProcCurveTo`, and `lpProcClosePath` are long pointers to the application-supplied callback functions that will be called by `ATMGetOutline()`.

The callback functions `ATMMoveToProc()`, `ATMLineToProc()`, `ATMCurveToProc()`, and `ATMClosePathProc()` correspond to the PostScript language operators `moveto`, `lineto`, `curveto`, and `closepath`, respectively. When passed to `ATMGetOutline()`, they correspond to the parameters passed to the PostScript language operator `pathforall`. These callback functions are not replacements for the older, 16-bit fill functions listed below. Outline filling can still be written into the application-supplied functions by calling Windows GDI fill functions or by calling these 16-bit functions. For 32-bit applications, it is recommended that the Windows GDI functions be used to fill the paths. In this case, the Bézier curve information will have to be converted to a data format supported by the Windows GDI.

As shown in the following callback prototype example, all the callback functions take similar arguments. Each callback function takes the `dwUserData` parameter and the appropriate number of pointers to `LPATMFixedPoint` parameters. The `ATMMoveToProc()` and `ATMLineToProc()` callbacks each take one `LPATMFixedPoint` parameter, and the `ATMCurveToProc()` callback takes three `LPATMFixedPoint` parameters; `ATMClosePathProc()` requires no `LPATMFixedPoint` parameters. Each of the `LPATMFixedPoint` parameters represents a device `(x, y)` coordinate or point (in the device context selected by `hDC`).
The *current point* can be set by using the `ATMMoveToProc()` callback with *device* coordinate \((x, y)\) pointed to by the `LPATMFixedPoint` parameter `lpFixPnt`.

A straight line segment can be appended to the current point using the `ATMLineToProc()` callback with *device* coordinate \((x, y)\) pointed to by the `LPATMFixedPoint` parameter `lpFixPnt`. The new current point will then be \((x, y)\).

A section of a Bézier cubic curve can be appended to the *current path* by using `ATMCurveToProc()`. Three *device* coordinates are needed, pointed to by the `LPATMFixedPoint` parameters `lpFixPnt1`, `lpFixPnt2`, and `lpFixPnt3`, which represent the points \((x1, y1)\), \((x2, y2)\), and \((x3, y3)\). A Bézier cubic section is added to the current path between the current point, referred to in Figure 6 as \((x_0, y_0)\), and the end 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.

**Figure 6** 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:

\[
\begin{align*}
x(t) &= ax^3 + bx^2 + cx + x_0 \\
y(t) &= ay^3 + by^2 + cy + y_0
\end{align*}
\]

The points returned by `ATMCurveToProc()` 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

\[
\begin{align*}
x_1 &= x_0 + cx/3 \\
y_1 &= y_0 + cy/3 \\
x_2 &= x_1 + (c_x + b_x)/3 \\
y_2 &= y_1 + (c_y + b_y)/3 \\
x_3 &= x_0 + c_x + b_x + a_x \\
y_3 &= y_0 + c_y + b_y + a_y
\end{align*}
\]

The current *subpath* can be closed by using `ATMClosePathProc()` to append a straight line segment connecting the current point to the starting point of the subpath.
The callback function prototypes should look something like this:

```c
BOOL ClosePath (DWORD dwUserData);
BOOL MoveTo (LPATMFixedPoint lpFixPnt, DWORD dwUserData);
BOOL LineTo (LPATMFixedPoint lpFixPnt, DWORD dwUserData);
BOOL CurveTo (LPATMFixedPoint lpFixPnt1,
              LPATMFixedPoint lpFixPnt2,
              LPATMFixedPoint lpFixPnt3,
              DWORD dwUserData);
```

where the `LPATMFixedPoint` arguments are the points passed by `ATMGetOutline()` to the callback functions and `dwUserData` is the message that the application program has `ATMGetOutline()` pass to the callback functions.

**Example 5:**

```c
... 
BOOL MyLineTo (LPATMFixedPoint lpFixPnt, DWORD dwUserData)
{
    /* manipulate and/or save the points lpFixPnt for use with GDI */
    /* functions or for use with the ATMFill functions. */
}

/* make 3 more functions as above for lineto, curveto & closepath */
...
Handle ghInst;
BOOL    result;
ATMFixedMatrix gMatrix = {ATMINTTOFIXED(100),0,0,
                            ATMINTTOFIXED(-100),0,0};
HDC     hDC;
HFONT   hFont;
BOOL    showResult;
ATMFixedMatrix matrix; /* may not be used */
char    myData[] = "Hello";
hWnd = CreateWindow(...);
    /* get a window handle*/
hDC = GetDC(hWnd)); /* get the device context to write to */

hFont = CreateFont(
    -1, /* font height, use -1 */
    0, 0, /* width, escapement */
    "Times"); /* Font name - a Type 1 font outline */

ATMSelectObject(..., ATM_Select, ...);
...
/* get proc instance address */
...
result = ATMGetOutline (hDC, 'A', &gMatrix, &MyMoveTo,
                   &MyLineTo,
                   &MyCurveTo,
                   &MyClosePath,
                   (DWORD) myData);
...

/* use the saved data */
```
Note that `ATMGetOutline()` returns paths in the same order in which the font designer created the character. This means that an application can not depend on information such as the outer path being returned before the inner path. In previous versions of the ATM API, the path returned by `ATMGetOutline()` did not incorporate any hints that might be in the font outline description. This has changed in the ATM 4.01 API. `ATMGetOutline()` now returns a hinted outline. Applications needing an unhinted outline should create a font at the font EM size (for example, 1000 pixels) and perform their own scaling. The `lpMatrix` parameter can not be used for this. Applications that perform their own scaling should always use a large font size; hinted outlines should not be scaled up in size.

A typical use of the data returned by `ATMGetOutline()` might be to have the application alter the data before sending the new data to the ATM software fill functions or to use the data to create some type of wire frame (that is, a 3-D font) and image the frame using Windows GDI functions.

`ATMGetOutline()` returns `ATM_BUSY` (see Section 7.1) if another callback has not returned.

**Note** For users of the 16-bit library, `ATMGetOutline()` can not be called in the middle of an `ATMFillStart()...ATMFillEnd()` block of operations.

### ATM 16-Bit Fill Functions

**Note** The following 16-bit functions are discouraged from use in the ATM 4.01 API: `ATMFillStart()`, `ATMFillMoveTo()`, `ATMFillLineTo()`, `ATMFillCurveTo()`, `ATMFillClosePath()`, `ATMFillGetBBox()`, `ATMFillEnd()`, and `ATMBaseFillEnd()`. These functions rely on preserving the state in ATM between function calls. This is not a reliable method in a Windows 95 multitasking environment. It is recommended that Windows GDI functions be used instead for filling arbitrary paths.

The 16-bit fill functions allow an application to use the ATM software to create and fill arbitrary paths. The ATM software fill functions `ATMFillStart()` and `ATMFillEnd()` must surround all calls to the path construction functions `ATMFillMoveTo()`, `ATMFillLineTo()`, `ATMFillCurveTo()`, `ATMFillClosePath()`, and `ATMFillGetBBox()`. These fill functions closely parallel the PostScript language operators `moveto`, `lineto`, `curveto`, `closepath`, and `eofill`, respectively.

The path is rendered and filled by the call `ATMFillEnd()` using the Windows GDI function `BitBlt()`. Therefore, like `BitBlit()`, the result will be determined by a combination of the pixels turned on by the fill operators, the `Brush` selected for the destination handle to the device context specified by `hDC`, the pixels previously turned on in the device context, the mapping mode
selected for the device context, and the raster operation specified by the parameter rop. See the Windows Programmer’s Reference included with the Windows SDK for information on the available raster operations.

**ATMFillStart**

`extern int ATMPI ATMFillStart (void);`

*Note: Version 1.0*

**ATMFillStart()** starts a new outline fill operation.

**ATMFillMoveTo**

`extern int ATMPI ATMFillMoveTo (LPATMFixedPoint lpFixPnt);`

*Note: Version 1.0*

**ATMFillMoveTo()** establishes a new current point without adding a segment to the current path.

The *lpFixPnt* parameter represents a device x, y coordinate. **ATMFillMoveTo()** starts a new subpath of the current path and sets the current point to (x, y).

*Note* **ATMFillMoveTo()** must be called at least once to establish a starting point before any calls to **ATMFillLineTo()** or **ATMFillCurveTo()** are executed.

**ATMFillLineTo**

`extern int ATMPI ATMFillLineTo (LPATMFixedPoint lpFixPnt);`

*Note: Version 1.0*

**ATMFillLineTo()** takes a pointer to *LPATMFixedPoint*, which represents a device x, y coordinate and appends a straight line segment from the current point to the point (x, y). The point (x, y) becomes the new current point.

The *lpFixPnt* parameter specifies the device x, y coordinates of the line endpoint.
**ATMFillCurveTo**  
extern int ATMAPI ATMFillCurveTo (  
    LPATMFixedPoint lpFixPnt1,  
    LPATMFixedPoint lpFixPnt2,  
    LPATMFixedPoint lpFixPnt3);  

*Note: Version 1.0*

**ATMFillCurveTo()** adds a section of a Bézier cubic curve to the current path. The parameters lpFixPnt1 and lpFixPnt2 represent the Bézier cubic control points. The parameter lpFixPnt3 represents the end point of the curve and the new current point. See the section above on **ATMGetOutline()** for more information on Bézier cubic curves.

**ATMFillClosePath**  
extern int ATMAPI ATMFillClosePath (void);  

*Note: Version 1.0*

**ATMFillClosePath()** closes the current subpath by appending a straight line segment connecting the *current point* to the subpath’s starting point.

**ATMFillEnd**  
extern int ATMAPI ATMFillEnd (  
    HDC hDC,  
    int x,  
    int y,  
    DWORD rop);  

*Note: Version 1.0*

**ATMFillEnd()** completes an outline fill operation by actually rendering the resulting image on a device specified by the *hDC* parameter and the raster operation denoted by the *rop* parameter.

The *hDC* parameter identifies the handle to the device context.

*x* and *y* specify the *logical* coordinates of the upper left corner of the image.

The *rop* parameter specifies the raster operation to be used. Any of the raster operations that can be used with the Windows GDI function **BitBlt()** can be used with **ATMFillEnd()**.

The inside of the current path is determined by the PostScript language *even-odd* rule. This is similar to the Windows GDI **ALTERNATE** fill mode. See the *PostScript Language Reference Manual, Second Edition* for details of the *even-odd* rule.

*Note*  
The image is drawn using the current text color specified in *hDC*. 
ATMBaseFillEnd extern int ATMAPI ATMBaseFillEnd (  
    HDC hDC,  
    int x,  
    int y,  
    DWORD rop);  

Note: Version 1.15  

ATMBaseFillEnd() is different from ATMFillEnd() only in the way that x and y are interpreted. In the case of this function, x and y are the coordinates of the start of the baseline, point (0,0), for the object being drawn and filled.

ATMFillGetBBox extern int ATMAPI ATMFillGetBBox (  
    HDC hDC,  
    LPATMBBox lpBBox);  

Note: Version 1.15  

ATMFillGetBBox() can be called before one of the FillEnd functions to obtain bounding box information for the outline being filled. The information obtained from this routine can help the caller to more accurately determine x and y coordinates for one of the FillEnd operations.

The hDC parameter is the device context handle.

lpBBox is a pointer to a structure which is a bounding box of the outline about to be filled (see Section 5). It is expressed in logical units and can be NULL. If NULL, then the bounding box is not returned.

Note This function can not be called after a call to either ATMFillEnd() or ATMBaseFillEnd().

Example 6:  

/* Produces output as in figure 7 */

...  

hDC = GetDC(hWnd); /* hWnd is the window to be drawn in */  
int x = 200, y = 200; /* Draw with origin at 200,200 */  
ATMFixedPoint p0, p1, p2;  
ATMFillStart();  

p0.x = (ATMFixed) (0);  
p0.y = (ATMFixed) (0);  
ATMFillMoveTo((LPATMFixedPoint) &p0);  

p0.x = (ATMFixed) (0);  
p0.y = ATMINTTOFIXED(200);  
ATMFillLineTo((LPATMFixedPoint) &p0);  

p0.x = ATMINTTOFIXED(50);  


p0.y = ATMINTTOFIXED(300);
p1.x = ATMINTTOFIXED(150);
p1.y = ATMINTTOFIXED(300);
p2.x = ATMINTTOFIXED(200);
p2.y = ATMINTTOFIXED(200);
ATMFillCurveTo((LPATMFixedPoint) &p0,(LPATMFixedPoint) &p1,
               (LPATMFixedPoint) &p2);

p0.x = ATMINTTOFIXED(200);
p0.y = (ATMFixed) (0);
ATMFillLineTo((LPATMFixedPoint) &p0);
ATMFillClosePath();
ATMFillEnd(hDC,x,y,SRCCOPY);

\*\*\*

\textbf{Figure 7} \textit{ATM Fill Functions}

In \textbf{Figure 7}, the gray-filled area was drawn by the code in \textbf{Example 6}. The points represent the line end points and the Bézier control points. The upper-left of the bounding rectangle is placed at \((x, y)\) as specified in the \texttt{ATMFillEnd()} function.
6.4 API TextOut Functions

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

*Note*  All API functions in this section work the same for both the 16-bit and 32-bit libraries, except where noted.

**ATMXYShowText**

```c
extern BOOL ATMAPP ATMXYShowText ( 
    HDC hDC,
    int x,
    int y,
    WORD wOptions,
    LPRECT lpRect,
    LPSTR lpString,
    int nCount,
    LPATMFixedPoint lpPoints,
    LPATMFixedMatrix lpMatrix);
```

*Note: Version 1.0*

**ATMXYShowText** renders text given an arbitrary transformation matrix, `lpMatrix`, and specific widths, `lpPoints`, for precise positioning of each character in the string. All but the last two parameters correspond to and are interpreted the same as the parameters of the Windows GDI function **ExtTextOut()**.

The `hDC` parameter identifies the handle to the device context.

`x` specifies the *logical* `x`-coordinate of the origin (upper left corner) of the character cell for the first character - either single-byte or double-byte - in the specified string.

`y` specifies the *logical* `y`-coordinate of the origin (upper left corner) of the character cell for the first character - either single-byte or double-byte - in the specified string.

*Note*  `x` and `y` are the coordinates of the upper left corner of the text box without taking the matrix into consideration. In other words, `(x, y)` defines the origin of a normal **TextOut()** or **ExtTextOut()** call which would use the font unmodified (i.e. with a default matrix). This means that the baseline position is determined relative to the unmodified font. Also note that `x` and `y` are expressed in logical units (just like the `x` and `y` parameters of the **TextOut()** and **ExtTextOut()** functions). However, the character widths passed in `lpPoints` are expressed in device units and are not subject to the current mapping mode in effect for `hDC`. This means that for devices that are capable
of rendering graphics at lower than normal resolutions (such as the PCL
driver’s ability to switch to 150 dpi or 75 dpi), these widths are expressed at
the graphics resolution in effect.

The wOptions parameter can be set to ETO_CLIPPED (although this value is
ignored), ATM_USELOGCOLOR, ATM_USEPURECOLOR,
ATM_MAKEVISIBLE or NULL (see Section 7.4). Opaque text is not
supported. That is, if wOptions includes the ETO_OPAQUE option, the option
will be ignored.

The lpRect parameter points to a data structure that contains the logical
coordinates of a rectangle. The lpRect parameter can be NULL. The effect of
the rectangle depends on how the wOptions parameter is set.

The lpString parameter points to the single- or double-byte character string to
be rendered.

nCount specifies the number of single-byte or double-byte characters in
lpString to be rendered.

The lpPoints parameter points to an array of x and y device coordinate values
that indicate the distance in x and y between the origins of adjacent single- or
double-byte characters. The lpPoints parameter can be NULL, in which case
the default character widths are used.

lpMatrix points to a transformation matrix of type ATMFixedMatrix (see
Sections 4 and 5). This transformation matrix will be applied to the string
being imaged. It is expressed in device units. The lpMatrix parameter can be
NULL. If it is NULL, then the font is used unmodified.

When creating a logical font for use with ATMXYShowText(), the lfHeight
parameter to the Windows GDI function CreateFont() should be specified
as –1. This allows ATM software to provide optimal placement when
displaying skewed or rotated text. This can be avoided by using
ATMBBoxBaseXYShowText().

ATMXYShowText() returns True if the call is successful; False otherwise.

Note For users of the 16-bit library, ATMXYShowText() can not be called in the
middle of an ATMFillStart()...ATMFillEnd() block of operations. In addition,
ATMXYShowText() has been included in the 32-bit library for compatibility
purposes, only. ATMBBoxBaseXYShowText() should be used instead.
ATMBBoxBaseXYShowText extern BOOL ATMAPI ATMBBoxBaseXYShowText (  
        HDC hDC,  
        int x,  
        int y,  
        WORD wOptions,  
        LPRECT lpRect,  
        LPSTR lpString,  
        int nCount,  
        LPATMFixedPoint lpPoints,  
        LPATMFixedMatrix lpMatrix,  
        BOOL bDoOutput,  
        LPATMBBox lpBBox,  
        LPATMFixedPoint lpDelta);  

Note: Version 1.15

ATMBBoxBaseXYShowText() is independent of, and the preferred function to, ATMXYShowText(). There are some slight differences that should be noted. x and y specify the coordinates of the start of the baseline rather than the upper left corner. Similar to ATMXYShowText(), x and y are expressed as logical units, lpRect is expressed in logical units, and lpPoints is expressed in device units.

If bDoOutput is True, then actual output is produced in the same manner as ATMXYShowText(). If only the font metrics are required, then bDoOutput should be set to False.

The lpBBox parameter will receive the bounding box (or extent) of the string. lpBBox is expressed in logical units and can be NULL. If it is NULL, then the bounding box is not returned.

The lpDelta parameter will receive the deltas that should be applied to x and y in order to make the subsequent call to this function continue at the correct point on the baseline. lpDelta is expressed in device units and can be NULL. If it is NULL, then the delta is not returned.

ATMBBoxBaseXYShowText() returns True if the call is successful; it returns False otherwise.

Note For users of the 16-bit library, ATMBBoxXYShowText() can not be called in the middle of an ATMFillStart()...ATMFillEnd() block of operations.
Example 7:

```c
/* Produces skewed text using ATMBBoxXYShowText() */

...#
#define DEGTORAD (2 * 3.14159/360) /* convert degrees to radians */
#define FIX(q) (ATMFixed) (q * 65536.0)
HDC hDC;
HFONT hFont;
BOOL showResult;
ATMFixedMatrix matrix;
static char someText[] = "Skewed";
int x = 100, y = 100; /* the start of the baseline*/
hWnd = CreateWindow(...); /* get a window handle */
hDC = GetDC(hWnd)); /* get the device context to write to */
hFont = CreateFont(-1, /* font height - use -1 with ATMXYShowText */
0, /* width */
0, /* escapement */
..., /* fill in the rest of the parameters */
"Times"); /* Font name - a Type 1 font outline */
ATMSelectObject(..., ATM_Select, ...);
matrix.a = FIX(44); /* 44 pixels in x */
matrix.b = FIX(-44*tan(15*DEGTORAD)); /* X axis 15 degrees
counter-clockwise */
matrix.c = FIX(73*tan(35*DEGTORAD)); /* Y axis 35 degrees
clockwise */
matrix.d = FIX(-73); /* 73 pixels in y */
matrix.tx = (ATMFixed)0;
matrix.ty = (ATMFixed)0;
showResult = ATMBBoxBaseXYShowText(hDC,x,y,NULL,(LPRECT) NULL,
someText, sizeof(someText)-1,
(LPATMFixedPoint) NULL,
&matrix, True, (LPATMBox) NULL, (LPATMFixedPoint) NULL);
```

Figure 8 ATMBBoxBaseXYShowText()
6.5 API Font Management Functions

This section contains information on functions that perform font management, such as adding and removing fonts on-the-fly, checking or changing the status of installed fonts, font change operations, and creating multiple master Font instances.

Note All API functions in this section work the same for both the 16-bit and 32-bit libraries, except where noted.

ATMBeginFontChange extern int ATMAPI ATMBeginFontChange (void);

Note: Version 2.5

ATMBeginFontChange() starts a series of font change operations consisting of calls to the functions ATMAAddFont() and ATMRemoveFont(). ATMBeginFontChange() must be called before starting a call or series of calls to ATMAAddFont() or ATMRemoveFont().

ATMFontStatus extern int ATMAPI ATMFontStatus ( 
    LPSTR lpMenuName, 
    WORD styleAndType, 
    LPSTR lpMetricsFile, 
    LPSTR lpFontFile);

Note: Version 2.5

ATMFontStatus() determines whether or not a given font is currently installed and/or in use by ATM. It may be called prior to installing a new font.

Note This function is not required before ATMAAddFont(). The details of ATMAAddFont() were changed in the ATM 4.01 API. For example, ATMFM (control panel) does not use ATMFontStatus().

The lpMenuName parameter contains the Windows name of the font.
The `styleAndType` parameter contains none, one, or more of the following flags (see Sections 5.5 and 5.6):

\[\text{ATM\_BOLD}\]
\[\text{ATM\_ITALIC}\]
\[\text{ATM\_TYPE1}\]
\[\text{ATM\_MMTYPE1}\]

**Note** The 2.5 version of ATM allowed only two style flags: `ATM\_BOLD` and `ATM\_ITALIC`.

The `lpMetricsFile` parameter contains the full pathname to the metrics file for the font. This is a .pfm file for an `ATM\_TYPE1` or a .mmm file for an `ATM\_MMTYPE1`.

The `lpFontFile` parameter contains the full pathname to the font file for the font. This is a .pfb file for both `ATM\_TYPE1` and `ATM\_MMTYPE1`.

Successful return values are as follows (see Section 7.1):

\[\text{ATM\_FONTINUSE}\]
\[\text{ATM\_FONTPRESENT}\]
\[\text{ATM\_FONTDIFFERENT}\]
\[\text{ATM\_FONTABSENT}\]

Failure return values are as follows:

\[\text{ATM\_PATHTOOLONG}\]
\[\text{ATM\_BADFONNTYPE}\]
\[\text{ATM\_NOFONT}\]
\[\text{ATM\_BADMENUNAME}\]
ATMAddFont extern int ATMAPI ATMAddFont (  
    LPSTR lpMenuName,  
    WORD  styleAndType,  
    LPSTR lpMetricsFile,  
    LPSTR lpFontFile);  

*Note: Version 2.5*

**ATMAddFont()** makes a newly installed font available to ATM.

The *lpMenuName* parameter contains the Windows name of the font.

The *styleAndType* parameter contains none, one, or more of the following flags (see Sections 5.5 and 5.6):

- **ATM_BOLD**
- **ATM_ITALIC**
- **ATM_TYPE1**
- **ATM_MMTYPE1**
- **ATM_TEMPORARY**

*Note The 2.5 version of ATM allowed only two style flags: ATM_BOLD and ATM_ITALIC.*

The *lpMetricsFile* parameter contains the full pathname to the metrics file for the font. This is a .pfm file for **ATM_TYPE1**. It is a .mmm file for an **ATM_MMTYPE1**.

The *lpFontFile* parameter contains the full pathname to the font file for the font. This is a .pfb file for both **ATM_TYPE1** and **ATM_MMTYPE1**.

The **ATM_TEMPORARY** bit may be set to add a font that is not enumerated.
The failure values are as follows (see Section 7.1):

- `ATM_INVALIDFONT`
- `ATM_PATHTOOLONG`
- `ATM_BADFONNTYPE`
- `ATM_NOROOM`
- `ATM_NOFONT`
- `ATM_BADMENUNAME`
- `ATM_MMMVERSION`

The `ATMRemoveFont` function is defined as follows:

```c
extern int A TMAPI ATMRemoveFont (
    LPSTR lpMenuName,
    WORD style);
```

Note: Version 2.5

`ATMRemoveFont()` makes a font inaccessible to ATM. Any subsequent requests to ATM for this font will fail. However, applications that have already selected this font into a `DC`, may continue to use it.

The `lpMenuName` parameter contains the Windows name of the font.

The `style` parameter contains none, one or both of the flags:

- `ATM_ITALIC`
- `ATM_BOLD`

The failure return value are as follows:

- `ATM_NOFONT`
- `ATM_BADMENUNAME`
ATMForceFontChange  extern int ATMAPI ATMForceFontChange(void);

*Note: Version 3.0*

**ATMForceFontChange()** forces a *WM_FONTCHANGE* message to be sent when **ATMEndFontChange()** is called regardless of whether or not a font was added.

ATMEndFontChange  extern int ATMAPI ATMEndFontChange(void);

*Note: Version 2.5*

**ATMEndFontChange()** finishes a series of font change operations consisting of calls to the functions **ATMAddFont()** and **ATMRemoveFont()**. This function must be called after font changes are completed. This function causes ATM to broadcast a *WM_FONTCHANGE* to all applications if any fonts have been added or removed. It also causes ATM to update its internal tables.

ATMMakePFM  extern int ATMAPI ATMMakePFM(
    LPSTR lpFaceName,
    WORD wStyles,
    HFILE hFile,
    LPSTR lpFileName);

*Note: Version 2.6*

Given a font name and style for a multiple master font instance, **ATMMakePFM()** constructs a .pfm file with blended metrics.

If *hFile* is equal to *HFILE_ERROR*, the function creates a temporary output file whose path is returned in *lpFileName*. In ATM 2.6 and 3.x, the file is created in the directory specified by ATM.INI, [Settings], TmpDir=. In ATM 4.0 and above, the Windows system call **GetTempFileName()** is used.

If *hFile* is not equal to *HFILE_ERROR*, it should be the handle of a file opened for writing. In this case, *lpFileName* is not used.

*Note*  For users of the 32-bit library, the *hFile* parameter must be equal to *HFILE_ERROR*.

The caller must close the file when *hFile* is not equal to *HFILE_ERROR*.
Failure return values are as follows (see Section 7.1):

- **ATM_MEMORY**
- **ATM_USEFILE**
- **ATM_CREATEFILE**
- **ATM_NOTFOUND**
- **ATM_SYSTEM**
- **ATM_DBERROR**
- **ATM_MMMVERSION**
- **ATM_NOMMMM**

```c
extern int ATMAPI A TMMakePSS (  
    LPSTR    lpFaceName,
    WORD     wStyles,
    HFILE    hFile,
    LPSTR    lpFileName);
```

**Note: Version 2.6**

Given a font name and style for a multiple master font instance, **ATMMakePSS()** constructs a .pss file for the PostScript driver. A .pss file is a stub file used only for PostScript printing.

If `hFile` is equal to **HFILE_ERROR**, the function creates a temporary output file whose path is returned in `*lpFileName`. In ATM 2.6 and 3.x, the file is created in the directory specified by ATM.INI, [Settings], TmpDir=, In ATM 4.0 and above, the Windows system call **GetTempFileName()** is used.

If `hFile` is not equal to **HFILE_ERROR**, it should be the handle of a file opened for writing. In this case, `lpFileName` is not used.

**Note**  For users of the 32-bit library, the `hFile` parameter must be equal to **HFILE_ERROR**.

The caller must close the file when `hFile` is not equal to **HFILE_ERROR**.
Failure return values are as follows (see Section 7.1):

\texttt{ATM\_RESOURCE}
\texttt{ATM\_USEFILE}
\texttt{ATM\_SYSTEM}
\texttt{ATM\_CREATEFILE}
\texttt{ATM\_MEMORY}
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 4.01 API. They are supported in both the 16-bit and 32-bit libraries, except where noted. 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 normal return value

ATM_INVALIDFONT -1
An invalid font error; the font is not consistent

ATM_CANTHAPPEN -2
An internal ATM error

ATM_BADMATRIX -3
An undefined inverse matrix

ATM_MEMORY -4
An out of memory error

ATM_NOTSUPPORTED -5
Wrong version of ATM for this call

ATM_NOTRUNNING -6
ATM is not currently running

ATM_FILLORDER -7
Inconsistent fill calls

Note: ATM_FILLORDER exists only in the 16-bit library.
ATM_CANCELLED -8
A client halted current operation

ATM_NOCHAR -9
No outline for character code in font

ATM_BADPROC -100
A bad callback function address

ATM_CANTDRAW -101
An error occurred in imaging; a bad raster operation(rop)

ATM_BADPARAM -102
A bad input parameter

ATM_SELECTED -200
See ATMSelectObject() for more information

ATM_NEWFONTSELECTED -200
See ATMSelectObject() for more information

ATM_OLDFONTSELECTED -201
See ATMSelectObject() for more information

ATM_FOREIGNFONTSELECTED -202
See ATMSelectObject() for more information

ATM_SELECTERROR -204
See ATMSelectObject() for more information
ATM_ERROR -219
A general error occurred, such as out of memory, or input string too long

ATM_NOTFOUND -223
The requested font was not found

ATM_NOFONTS -224
No fonts were enumerated

ATM_SYSTEM -226
An operating system call error

ATM_BUSY -227
This reentry of the library requires a resource that is still in use

ATM_PATHTOOLOOKG -400
At least one path in the current file name is too long

ATM_BADFONNTYPE -401
The font is uncompressed

ATM_BADSTYLE -402
A style with bits set other than ATM_ITALIC or ATM_BOLD

ATM_NOROOM -403
Can not add to internal tables

ATM_NOFONT -404
At least one path does not point to the correct file

ATM_BADMENUNAME -405
Missing menu name or invalid menu name given
ATM_FONTINUSE -406
The paths and menu name specify an existing font that is in use

ATM_FONTPRESENT -407
The paths and menu name specify an existing font which is NOT in use

ATM_FONTDIFFERENT -408
The font, as specified by its menu name and styles, exists but its data files are
different from those specified by the lpMetricsFile and lpFontFile parameters

ATM_FONTABSENT -409
There is no such font installed

ATM_ADDERROR -410
A general ATMAddFont() error

ATM_CREATEFILE -415
Could not create file

ATM_USEFILE -416
Could not read, write, or close file

ATM_ADDDELAYED -417
The current add is incomplete, pending file access

ATM_MMMISSING -418
Add and/or FontStatus could not process instance because base font is not
installed

ATM_NOMMMFORMM -419
An attempt was made to add an multiple master base font with a .pfm/.pfb
pair; a .mmm type file is required for this
7.2 Return values for the font substitution database

ATM_ALREADYINSTALLED -2000
The requested font was already installed

ATM_NOFONTFILE -2001
Bad or missing .pfb file

ATM_NOMETRICSFILE -2002
Bad or missing .pfm or .mmm file

7.3 Return values for PostScript stub and .pfm generation

ATM_BADPSNAME -2100
An invalid PostScript font name

ATM_NODATABASE -2101
No font substitution database was found at ATM start-up, or ATM was unable to initialize the font substitution database

ATM_DBERROR -2102
A font substitution database error, typically a corrupt or malformed font substitution database

ATM_NOMM -2200
The font is not an multiple master instance or a substitute

ATM_MMMVERSION -2201
This is an old version of the .mmm file

ATMRESOURCE -2202
A resource error
7.4 Additional flags for the wOptions parameter

These define additional flags for use with the wOptions parameter of the ATMXYShowText() and ATMBBoxBaseXYShowText() functions. These flags are not supported in all versions of the ATM API. Check the version for each flag.

**ATM_USELOGCOLOR** 0x0100
In the ATM API since version 2.01. This flag causes ATM to use the logical color set for text foreground in the device context (DC) even when it would normally switch to the nearest pure color based on the settings in ATM.INI. This is mutually exclusive with **ATM_USEPURECOLOR**. Incorrect usage will cause an error.

**ATM_USEPURECOLOR** 0x0200
In the ATM API since version 2.01. This flag causes ATM to use the nearest pure color instead of the actual logical color set for text foreground in the DC. This is mutually exclusive with **ATM_USELOGCOLOR**. Incorrect usage will cause an error.

**ATM_MAKEVISIBLE** 0x0400
In the ATM API since version 2.01. This flag is used for monochrome devices and causes ATM to use either white or black depending on the color of background in order to make the text visible. If this flag is specified, **ATM_USELOGCOLOR** and **ATM_USEPURECOLOR** are ignored.

7.5 Additional flags for the options parameter

These define additional flags for use with the options parameter of the ATMSSelectObject() routine. These flags are not supported in all versions of the ATM API. Check the version for each flag.
ATM_DEFER 0x0001
In the ATM API since version 2.01. This flag causes ATM to defer to a device font when selecting a font handle into a DC the first time. This is mutually exclusive with ATM_SELECT. Incorrect usage will cause an error.

ATM_SELECT 0x0002
In the ATM API since version 2.01. This flag forces ATM to render a font even when normally it would be deferred to the device. This can happen only when selecting a handle into a DC the first time. This is mutually exclusive with ATM_DEFER. Incorrect usage will cause an error.

ATM_USEEXACTWIDTH 0x0004
In the ATM API since version 2.01. This flag causes ATM to create the font being selected into a screen DC without adjusting the width data. This can happen only when selecting a handle into a DC for the first time.

ATM_ITALIC 0x0001
In the ATM API since version 2.6. This flag is used in calls to and return values from ATM to indicate the style of a font’s face name. If set, it indicates that the style associated with the font is ITALIC.

ATM_BOLD 0x0002
In the ATM API since version 2.6. This flag is used in calls to and return values from ATM to indicate the style of a font’s face name. If set, it indicates that the style associated with the font is BOLD.

ATM_BOLDITALIC ATM_BOLD | ATM_ITALIC

7.6 Type bits for ATMEnumFonts() and ATMAddFonts()

ATM_TYPE1 0x0100
A normal Type 1 .pfb file

ATM_MMTYPE1 0x0200
A multiple master Type 1 .pfb file
**ATM_MMINSTANCE** 0x0400
An instance of multiple master font

**ATM_DATABASE** 0x0800
A database substitution font

**ATM_TEMPORARY** 0x1000
Can be logically OR-ed with the other bits

**ATM_PSNAMESIZE** 64
max length for a PostScript font name

**ATM_MAXPATHLEN** 260
max length for .pfb and .pfm paths

### 7.7 mmFlags for the ATMMMetricsHeader structure

These flags are passed by the `mmFlags` parameter defined in the `ATMMMetricsHeader` structure used by `ATMEnumMMFont()`.

*Note: see Section 5 for the declaration of the ATMMMetricsHeader structure.*
**MM_SUBSTOK** 0x01
The font is usable for substitution

**MM_TEXTINSTOK** 0x02
The font is usable for text instances

**MM_IS_PI** 0x04
The font is decorative

**MM_IS_FIXED_PITCH** 0x08
The font is fixed pitch

**ATM_MAXAXES** 4
The maximum number of axes for a multiple master font

**ATM_MAXMASTERS** 16
The maximum number of master designs for a multiple master font

### 7.8 mmAxisAttrs for ATMMMetricsHeader structure

These flags are passed by the `mmAxisAttrs` parameter defined in the `ATMMMetricsHeader` structure used by `ATMEnumMMFont()`.

*Note: see Section 5 for the declaration of the `ATMMMetricsHeader` structure.*
7.9 Various flags supported by ATMSetFlags()

These flags are passed by the flags parameter of ATMSetFlags().

Note: See Section 6.6 for more information on ATMSetFlags().

ATM_DOWNLOAD 0x0001
Enable soft fonts for printer drivers

ATM_USEDEVFONTS 0x0002
Defer to device fonts

Note: The following flags were added in the ATM API, version 4.0.
**ATM_SUBSTITUTE** 0x0004
Enable font substitution

**ATM_ANTIALIAS** 0x0008
Enable anti-aliasing for the screen

**ATM_AUTOACTIVATE** 0x0010
Enable automatic font activation

**ATM_GDIFONTS** 0x0020
Enable Windows GDI fonts for screen drivers

### 7.10 Return values for *lpFromOutline in ATMFontAvailable()*

The following integer return values are passed by the *lpFromOutline parameter of ATMFontAvailable().

*Note: See Section 6.1 for more information on ATMFontAvailable().*

**ATM_SYNTH** 0
The font style may be synthesized

**ATM_REAL** 1
The exact font and style are available

**ATM_SUBST** 2
The substitute font and style are available
Appendix A: Changes Since Earlier Versions

Changes effective January 24, 1997

• Added more complete information to the introduction, Sections 1 and 2.

• Added support for double-byte characters.

• Updated the index to include the new API functions, parameters, and flags.

• Updated the introduction to include the new features of ATM 4.0, ATM 4.0 Deluxe, and the API version 4.01.

• Updated the information on the Transformation Matrix in Section 1.

• Added a List of Figures

• Added a List of Examples

• Dropped the old Appendix A (Error Codes) and moved the information to Section 2.

• Dropped the old Appendix B (More on Transformation Matrices) and moved the information to Section 1.

• Created Sections 2, 3, and 4.

• Added many new Return Values and Flags in Section 2.

• Updated the Structures and Data Types in Section 3.

• Reorganized the list of API calls in Section 4. Created the 6 new categories of calls.

• Added several new API calls (functions), including ATMGetBuildStr(), ATMFinish(), ATMMakePFM(), ATMMakePSS(), ATMSetFlags(), ATMEnumFonts(), ATMEnumMMFonts(), ATMEnumFonts(), ATMGetMenuName(), ATMGetPostScriptName(), ATMSelectObject(), and ATMForceFontChange().
• Updated and simplified Appendix B, Obsolete API Functions.

Changes since April 14, 1993

• Added more descriptions relating to ATM software for Windows Japanese version: Section 2.2 under \texttt{ATMXYShowText()}. 

Changes since March 1, 1993

• Added descriptions relating to ATM software for Windows Japanese version.
• Added a new function \texttt{ATMGetOutline()}. 

Changes since February 14, 1992

• Update to the note on the \texttt{ATMBox} structure.
• Changes to \texttt{ATMXYShowText()} and \texttt{ATMBoundingBoxXYShowText()}. 
• Added new functions: \texttt{ATMSelectObject()}, \texttt{ATMAddFont()}, \texttt{ATMRemoveFont()}, \texttt{ATMBeginFontChange()}, \texttt{ATMEndFontChange()}, and \texttt{ATMFontStatus()}. 
• Appendix D, “Obsolete API Functions” was added. 

Changes since August 9, 1991

• The document name changed from “Advanced Type Capabilities Using Adobe Type Manager Software For Windows” to “Adobe Type Manager Software API: Windows”.
• The document now reflects various changes to the API as of ATM version 2.0. In addition to some bug fixes, two API functions were added. These include: \texttt{ATMForceDefer()}, and \texttt{ATMClearAllForce()}. 
• The format of the document was updated. 
• Various minor typographical errors were corrected.
Appendix B: Obsolete ATM API Functions

The following ATM API functions are considered obsolete. Even though they are all listed and described in Section 6, they are discouraged from being used. A replacement function is listed below for each. Each of these replacement functions are also fully described in Section 6. It is strongly recommended that these replacement functions be used.

Use the ATM API function `ATMSelectObject()` as a replacement for the following obsolete calls:

- `ATMClearAllForce()`
- `ATMForceCreate()`
- `ATMForceDefer()`
- `ATMForceExactWidth()`

Use Windows GDI functions as a replacement for the following obsolete calls:

- `ATMFillStart()`
- `ATMFillMoveTo()`
- `ATMFillLineTo()`
- `ATMFillCurveTo()`
- `ATMFillClosePath()`
- `ATMFillGetBBox()`
- `ATMFillEnd()`
- `ATMBaseFillEnd()`
Symbols

.mmm 17
.pfb 17
.pfm 17
.pss 17

Numerics

16-bit 10, 16, 18, 25, 31, 38, 40, 41, 42, 46, 51, 52, 53, 55, 62
32-bit 10, 15, 16, 17, 18, 25, 31, 32, 38, 43, 51, 52, 55, 59, 60, 62

A

Adobe PostScript driver 14
anti-aliased text 9, 15
anti-aliasing 72
ATM Control Panel 18, 38, 39
ATM.INI 14, 34, 37, 59, 60, 67
ATMAddFont 12, 18, 30, 55, 57, 59, 65, 68
ATMBaseFillEnd 18, 29, 46, 49
ATMBox 25
ATMBBoxBaseXYShowText 18, 24, 29, 52, 53, 54, 67
ATMBeginFontChange 30, 55
ATMClearAllForce 17, 29, 38, 40
ATMClosePathProc 27, 43, 44
ATMCurveToProc 28, 43, 44
ATMEndFontChange 18, 30, 59
ATMEnumFontProc 26, 32
ATMEnumFonts 17, 29, 32, 68
ATMEnumMMFont 69
ATMEnumMMFontProc 27, 32
ATMEnumMMFonts 17, 29, 32
ATMEnumMMInstInfo 27, 32
ATMFillClosePath 18, 29, 46, 48, 50
ATMFillCurveto 18, 29, 46, 47, 48, 50
ATMFillCurveTo 18, 29, 46, 47, 48, 50
ATMFillEnd 18, 29, 46, 48, 49, 50, 52, 53
ATMFillGetBBox 18, 29, 46, 49
ATMFillLineTo 18, 29, 46, 47, 49, 50
ATMFillMoveTo 18, 29, 46, 47, 49
ATMFillStart 18, 29, 46, 47, 49, 52, 53
ATMF Finish 17, 29, 31
ATMFixed 22, 25, 54
ATMFixedMatrix 18, 19, 25, 45, 52, 54
ATMFixedPoint 25, 49
ATMFontAvailable 12, 13, 17, 29, 33, 34, 72
ATMFontPathInfo 27
ATMFon tSelected 29, 34
ATMFon tSpec 26
ATMFon tStatus 30, 55
ATMForceCreate 17, 29, 38, 40, 41
ATMForceDefer 17, 29, 38, 40, 41
ATMForceExactWidth 17, 29, 38, 40, 41
ATMForceFontChange 18, 30, 59
ATMGetBuildStr 17, 29, 37
ATMGetFontBBBox 25, 29, 34
ATMGetFontBBox 14, 15, 17, 29, 35
ATMGetMenuName 17, 29, 35, 36
ATMGetOutline 18, 24, 27, 28, 29, 42, 43, 45, 46, 48
ATMGetOutline2 18, 42
ATMGetOutlineName 17, 29, 36
ATMGet Version 29, 36
ATMINTTOFIXED 21, 25, 45, 49, 50
ATMLineToProc 28, 43, 44
ATMMakePFM 18, 30, 59
ATMMakePSS 18, 30, 60
ATMMMMetricsHeader 26, 32, 69, 70
ATMMoveToProc 27, 43, 44
ATMProperlyLoaded 29, 31
ATMRemoveFont 30, 55, 58, 59
ATMSelectObject 11, 15, 17, 18, 29, 38, 39, 40, 41, 45, 54, 63, 67
ATMSetFlags 15, 18, 29, 37, 71
ATMShortPoint 25
ATMXYShowText 18, 20, 24, 29, 43, 51, 52, 53, 54, 67
Auto-activated 14
Auto-activation 11
auto-activation 9, 11, 12, 13, 15, 72

B
Bézier 10, 43, 44, 48, 50
BitBlt 46, 48

C
closepath 42, 43, 46
CreateFont 33, 40, 41, 45, 52, 54
CreateFontIndirect 40, 41
CTM 23, 24
Current Transformation Matrix 11, 23
curveto 42, 43, 46

D
Device Context 11
device context 15
device space 23, 42
DLL file 10, 11, 16
DLL files 17, 37
double-byte 9, 16, 25, 42, 43, 51, 52, 62
DOWNLOADFACE 14

E
eofill 46
Escape 15
export font sets 9
exported font sets 16
ExtEscape 15
ExtTextOut 39, 51

F
FillEnd 49
findfont 15
font enumeration 9
font management 9, 10, 11
font sets 9, 16
font smoothing 9, 11, 15
Font substitution 12
font substitution 9, 11, 12, 13, 14, 15, 35, 36, 66, 72
font substitution database 66

G
GDI 9, 18, 33, 36, 39, 43, 45, 46, 48, 51, 52, 72
GetCharABCWidths 15
GetCharWidth 15
GETEXTENDEDTEXTMETRICS 15
GETFACENAME 14
GetFontData 15
GetGlyphOutline 15
GetKerningPairs 15
GetOutlineTextMetrics 15
GETPAIRKERNTABLE 15
GetTextMetrics 15
GetVersion 36

J
Japanese fonts 9

K
kanji 10

L
lineto 42, 43, 46
lpFromOutline 12, 13, 33, 72

M
makefont 19
Microsoft PostScript driver 14
Microsoft Windows 9
Microsoft Windows SDK 14
MM_TEXT 25
MMM 35
mmm 56, 57, 65, 66
moveto 42, 43, 46
multiple master font 12
multiple master fonts 9, 12, 18
multiple master instances 10, 11, 12, 13, 14, 15, 17, 32

P
PASSTHROUGH 14
pathforall 42, 43
PFB 35
pfb 56, 57, 65, 66, 68, 69
PFM 35
pfm 18, 56, 59, 65, 66, 69
PostScript 9, 10, 11, 12, 14, 15, 17, 18, 19, 23, 24, 32, 34, 35, 36, 42, 43, 46, 48, 60, 66, 69
PostScript driver 15, 18, 34, 60
PostScript font 17, 66, 69
pss 18, 60

R
rotate 23

S
scale 23
SelectObject 11, 38, 40, 41
single-byte 9, 16, 25, 42, 51, 52, 62

T
TextOut 13, 14, 29, 39, 43, 51
translate 23
TrueType 9, 14, 15, 16
Type 1 9, 12, 14, 15, 16, 17, 18, 32, 33, 34, 39, 42, 45, 54, 68

W
WIN.INI 14
Windows 9, 10, 11, 12, 17, 18, 23, 33, 35, 36, 38, 39, 43, 46, 47, 48, 51, 52, 55, 57, 58, 59, 60, 72
Windows 3.1 9, 10, 16
Windows 95 9, 10, 11, 12, 14, 15, 16, 46
Windows GDI 10, 11, 13, 14, 15, 16
WM_FONTCHANGE 18, 59