Adobe Type Manager Software API With Multiple Master Fonts: Macintosh

Adobe Developer Support

Technical Note #5074

14 February 1992

Adobe Systems Incorporated
Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/
Copyright © 1992 by Adobe Systems Incorporated. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher. Any software referred to herein is furnished under license and may only be used or copied in accordance with the terms of such license.

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

Any references to a “PostScript printer,” a “PostScript file,” or a “PostScript driver” refer to printers, files, and driver programs (respectively) which are written in or support the PostScript language. The sentences in this book that use “PostScript language” as an adjective phrase are so constructed to reinforce that the name refers to the standard language definition as set forth by Adobe Systems Incorporated.

PostScript, the PostScript logo, Adobe, the Adobe logo, Adobe Type Manager, ATM, and Myriad are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. Macintosh is a registered trademark of Apple Computer, Inc. Other brand or product names are the trademarks or registered trademarks of their respective holders.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third party rights.
Contents

Adobe Type Manager Software API for Multiple Master Fonts: Macintosh 5
1 Introduction to Multiple Master Fonts 5
   Design Space Examples 6
2 API Related Technical Overview 7
   Basic Terminology 7
   Font Names 8
   Miscellaneous 11
3 ATM Software API Routines 13

Appendix A: The makeblendedfont Operator 35
Appendix B: 5-3-3 Rule for Macintosh File Names 37
Appendix C: ATM software Error Codes 39
Appendix D: Function Prototype Summary 41
Appendix E: Font Axis and Label Names 45
Appendix F: Supporting EPS Files 47
Index 53
1 Introduction to Multiple Master Fonts

A multiple master typeface incorporates from one to four design axes of varying dynamic ranges. The typeface designer or manufacturer determines the number of design axes a multiple master typeface should have and the range of each axis, depending on what is appropriate for the typeface.

A simple multiple master typeface might have a single design axis for weight, with a range from light to bold. A more complex multiple master typeface might incorporate two design axes, for example, one for weight, with a range from extra light to black and one for width, with a range from condensed to expanded.

Table 1 shows examples of design axes and dynamic ranges for multiple master typefaces.

Table 1 Design axes and dynamic ranges for multiple master typefaces

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Design axis</th>
<th>Dynamic range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-axis typeface</td>
<td>Weight</td>
<td>light to black</td>
</tr>
<tr>
<td>2-axis typeface</td>
<td>Weight, Width</td>
<td>light to black, condensed to expanded</td>
</tr>
<tr>
<td>3-axis typeface</td>
<td>Weight, Width, Optical size</td>
<td>light to black, condensed to expanded, 6 pt to 72 pt</td>
</tr>
<tr>
<td>4-axis typeface</td>
<td>Weight, Width, Optical size, Contrast</td>
<td>light to black, condensed to expanded, 6 pt to 72 pt, high to medium contrast</td>
</tr>
</tbody>
</table>
1.1 Design Space Examples

Figures 1 through 4 are examples of design space.

**Figure 1** One design axis – 2 masters

```
<table>
<thead>
<tr>
<th>Light</th>
<th>Black</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>design axis 1: weight</td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 2** Two design axes – 4 masters

```
<table>
<thead>
<tr>
<th>Light</th>
<th>Expanded</th>
<th>Black</th>
<th>Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>design axis 1: weight</td>
<td>design axis 2: width</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light</td>
<td>Black</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 3** Three design axes – 8 masters

```
<table>
<thead>
<tr>
<th>Light</th>
<th>Expanded 72 pt</th>
<th>Black</th>
<th>Expanded 72 pt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>design axis 1: weight</td>
<td>design axis 2: width</td>
<td>design axis 3: size</td>
<td></td>
</tr>
<tr>
<td>Light</td>
<td>Condensed 72 pt</td>
<td>Black</td>
<td>Condensed 72 pt</td>
</tr>
<tr>
<td></td>
<td>Expanded 6 pt</td>
<td>Black</td>
<td>Expanded 6 pt</td>
</tr>
<tr>
<td>Light</td>
<td>Condensed 6 pt</td>
<td>Black</td>
<td>Condensed 6 pt</td>
</tr>
</tbody>
</table>
```


2 API Related Technical Overview

This section briefly discusses multiple master concepts and miscellaneous information that corresponds to information an application must consider when using the ATM™ API with multiple master fonts.

2.1 Basic Terminology

A multiple master font program contains two or more typeface designs called master designs which describe the design axes that define a multiple master font. Table 1 shows some example design axes along with typical dynamic ranges. It is possible to think of master designs as being arranged in a 1, 2, 3, or 4 dimensional design space with various instances of the font corresponding to different locations in that space. Figures 1 through 4 illustrate the design spaces that correspond to the examples in Table 1.

A blended font (also known as a font instance) corresponds to a selected point in the design space of the multiple master font. The result is a weighted average of the master designs in the font. In addition to the master designs in the corners of the design space, there can be interior designs that may impose a non-linear quality to the interpolation.

Note Interior designs are not supported in ATM 3.0.

Currently, the maximum number of master designs allowed in a blended font is 16 and will be \(2^n + x\), where \(n\) is the number of axes and \(x\) is the number of interior designs. In a blend of \(k = 2^n + x\) master designs, there are \(k\) weights that sum to 1.0 and determine the relative contributions of the different designs. The weights are stored in the PostScript™ font dictionary in an array named WeightVector.
An example of a weight vector might be: [.25 .25 .1 .4]. This data is maintained as the variable `weightVector` in this document. The weight vector can also be used to calculate other useful information on the host. See `fontFit()` for an example.

User design coordinates represent values within the dynamic range of an axis. For example, in figure 3 above, the user design coordinates of the optical size axis can range from 6 to 72 inclusive. In this document and in the ATM API, the values at the extremes of the user design coordinates are referred to as `userMin` and `userMax` (e.g. `userMin = 6` and `userMax = 72`) and the user design coordinates are maintained as an array of `Fixed` values pointed to by the variable `*coords`. The contents of `coords` array for figure 3 if all the axes were set in the middle, would be: 500, 500, and 36 if Light to Black and Condensed to Expanded both ranged from 0 to 1000, however, the actual values in the array would be represented as type `Fixed`.

Normalized design coordinates are a mapping from user design coordinates so that all the values range from 0.0 to 1.0. The normalized design coordinates are represented in this document as an array of `Fixed` values in the variable pointed to by `*normalCoords`. Normalized coordinates are convenient to use for mathematical manipulations.

Primary fonts are a set of a multiple master font instances that have been “blessed” by the font designer as being good choices for light, bold, condensed, expanded, regular, etc. These primary fonts are also known as blessed fonts in this document and in the API.

Custom fonts are multiple master font instances that are not Primary fonts.

A multiple master font name is composed of two parts. The first part is the family name and the second is the display instance. The display instance (also known as the display string) is a string of numbers and labels that specify where in design space the font lies. It is the portion following the underscore character in the font name. In the following example, the bold characters represent the display instance: MyriaMM_400 RG 200 CN.

Regular instance is the font instance that is determined by the designer to be the “regular style” of the font.

2.2 Font Names

This section discusses two formats of multiple master font names. The first being the Macintosh® FOND names, those names that will show up in font menus which are built via `AddResMenu()`. These names will also be referred to in this document as “Macintosh font names”. Secondly, the PostScript font names are discussed. These names will show up in menus created via parsing the FOND style mapping tables. These names will also be found in Post-
Script Language files. For a more complete description of naming conventions, see Technical Note #5087, *Multiple Master Font Programs For The Macintosh*.

**FOND Names**

Macintosh FOND names are defined as follows:

\(<\text{Family Name [MM]}\>\ [\text{Style/Char Set}_1\ldots\text{Number}_n\ \text{Label}_n\)

where items in brackets are optional.

*FamilyNameMM*

Required. Identifies the family name of the typeface family, such as *Myriad*. The *MM* is not required, but will help to identify multiple master typefaces. Because of the constraints on the length of font names, the family name for Adobe™ fonts is currently derived by applying the 5-3-3 rule to the PostScript font name. See Appendix B for details of the 5-3-3 rule.

For example, the PostScript font family name *MyriadMM* will have the FOND family name *MyriaMM*, which is also the Macintosh font file name.

*Style/Char Set*

Optional. Specifies style characteristics such as *italic*, and/or character set information such as *Expert* for the Expert character set. For example, in the font name MyriaMMIta_932 wt 500 wd, MyriaMMIta represents MyriadMM-Italic.

* - (underscore character)

Required. The family name/style/char set portion of the name and the design coordinate/label portion must be separated by an underscore character.

*Number*

Required. This represents the design coordinate number specified by the designer. For a font with *n* axes, there will be *n* numbers. The numbers must be separated by one or more non-numeric characters: text labels are recommended if space permits, otherwise the use of a space or underscore character is recommended.

*Label*

The axis Label field is required, though it may optionally consist of only a single non-numeric character to separate the design coordinates. However, descriptive abbreviation labels are recommended. Labels for primary fonts will have upper case letters and labels for custom fonts will be in lowercase letters. See Appendix E for more information on labels.
Example 1: Examples of Font Names

MyriaMM_450 RG 600 NO
MyriaMMIta_932 wt 500 wd

In this document, the Number/Label combination is referred to as the display instance.

Font Name Length

Font names may be up to 31 characters in length, with the first 28 characters being unique and characters 29 through 31 being extra label information only.

PostScript Language Font Names

The PostScript language font names are strings that do not contain spaces. These names contain an underscore connecting the sub-strings of the display instance. For example, the Macintosh name MyriaMMIta_450 RG 600 NO will map (via the FONDs style mapping table) to the PostScript language font name MyriadMM–Italic_450_RG_600_NO.

If it is necessary to convert a PostScript font name to a Macintosh name that can be used with the API functions and with GetFNum(), use a combination of decodeBlendedFontNameATM() and encodeBlendedFontNameATM() as described in section 3, “ATM Software API Routines”.

Unless otherwise specified, when the words “font name” are used in this document, they refer to the Macintosh font name as described in the above section “FOND Names” and not to the PostScript font name.

Patch to GetFNum()

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

ATM versions 3.0 and higher patch GetFNum() so that if the font name is a multiple master blend name containing a display instance that does not currently exist, ATM will create the instance (assuming the Master font exists) for the application in ATM’s temporary suitcase. The temporary suitcase will remain until the system is re-booted (see the section, “The ATM Temporary Suitcase”).

Carry The Font Names In The Document

Fonts should be stored and identified by name in a document, not by number. Please see Apple DTS Technical Note # 191 for a suggested font strategy.
When opening or including a document and looking for the fonts, if the font is not found in the system, don’t stop there! Do a `GetFNum()` on the name and ATM will create the font if it is a multiple master instance name whose master fonts are available on the system. Finally, update the font menu and any relevant tables and internal structures.

**The ATM Temporary Suitcase**

ATM stores temporary instances of multiple master fonts in a suitcase called `ATMTemp` in the system’s `Preferences` folder. This suitcase can be used by an application to store both temporary and permanent fonts, however, even permanent fonts will be removed from this suitcase on re-boot.

**2.3 Miscellaneous**

**So Many Functions**

There are over twenty new API functions for ATM for working with multiple master fonts. This may seem overwhelming at first glance; however, several of the API functions documented here will never have to be used by a typical application. These are functions used by ATM and the Font Creator which have been brought out to the API in order to not limit any application’s functionality. For example, `convertCoordsToBlendATM()` is intended for use by applications that output their own PostScript language code. `createTempBlendedFontATM()` and `disposeTempBlendedFontATM()` may rarely be used by applications since `showTextDesignATM()` is more efficient in most situations. `convertNormToUser()` and `convertUserToNorm()` are for an application that needs to do calculations on the coordinate values. `createPermBlendedFontATM()`, `disposePermBlendedFontATM()`, `getBlessedFontNameATM()`, and `getNumBlessedFontsATM()` are used by the Font Creator/Picker, in fact, `MMFontPickerATM()` will do the work of several of the API functions.

**The makeblendedfont Operator**

The PostScript language operator `makeblendedfont` may be used by applications that output PostScript language code and is documented in Appendix A.

**Why An Application Might Use The API**

If a multiple master font family has been installed into the system or into an open suitcase, an application’s call to the toolbox function `AddResMenu()`, will add a set of `primary` font instances to the application’s font menu and these fonts will be available like any other font. If an application’s designer wishes to extend the font capabilities to take full advantage of the multiple master technology, the application can use the ATM API. Through the API,
the application has access to over 20 multiple master related functions that help give an application abilities such as easily creating and deleting custom font instances, providing copy fit options, and even provide font matching capabilities. Following are some scenarios of basic API usage which will illustrate how easy extending multiple master functionality can be.

Scenario: The user wants to create some new custom font instances, ones that were not distributed as primary fonts with the multiple master font package.

The application calls MMFontPickerATM( ) in “creator” mode which allows the user to navigate through the design space of a multiple master font and display sample strings of various font instances. When the user selects a desired instance(s) by hitting the “Create” button, a permanent font is created for the user. The font(s) can then be added to the applications font menu via the toolbox’s InsMenuItem( ) or AppendMenu( ) functions.

MMFontPickerATM( ) can also be used to remove previously created font instances.

Functions to Get Started With

1: Basic multiple master support that allows the user to create a font instance without leaving the application: MMFontPickerATM( ).

2: Convert a PostScript font name into a name suitable for use with GetFNum( ) and API functions: encodeBlendedFontNameATM( ).

3: EPS support: decodeBlendedFontNameATM( ), and getFontFamilyFONDATM( ).

4: Copy fitting capabilities: copyFitATM( ),getNumMastersATM( ), getMasterFONDATM( ), and showTextDesignATM( ).
3 ATM Software API Routines

It is expected that Technical Note #5072, Adobe Type Manager™ Software API: Macintosh be used in conjunction with this document. Technical Note #5072 describes how to initialize the ATM API, has a listing of ATM error codes, and describes how to do text manipulation with a transformation matrix.

Note Several of the API routines in this document expect the current font to be a multiple master font. If one of these calls is made and the current font is not a multiple master font, the error code ATM_NOT_BLENDED_FONT is returned.

addMacStyleToCoordsATM

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

addMacStyleToCoordsATM() allows an application to add a style value to a set of coordinates.

When a style bit is set on a multiple master typeface, ATM software adds a calculated amount (determined by the font designer) to the coordinates of the axis corresponding to the style, if one exists. For example, if the bold bit is set, then the coordinates of the weight axis increase. This style value can be retrieved by addMacStyleToCoordsATM() and used via the API.

coords is an array of Fixed user design coordinate values. coords and newCoords can be pointers to the same array.

macStyle is a short whose bit positions have the following meanings:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldBit</td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td>italicBit</td>
<td>= 1</td>
<td></td>
</tr>
<tr>
<td>underlineBit</td>
<td>= 2</td>
<td></td>
</tr>
<tr>
<td>outlineBit</td>
<td>= 3</td>
<td></td>
</tr>
<tr>
<td>shadowBit</td>
<td>= 4</td>
<td></td>
</tr>
<tr>
<td>condenseBit</td>
<td>= 5</td>
<td></td>
</tr>
<tr>
<td>extendedBit</td>
<td>= 6</td>
<td></td>
</tr>
</tbody>
</table>

newCoords returns the new user coordinates incremented by the value that corresponds to the given macStyle. *coords and *newCoords can be pointers to the same array.
stylesLeft returns the styles for which the blended font has no corresponding axis. For example, if FF (11111111) is passed in macStyle, and the font has a weight axis that corresponds to the bold bit and no other axes corresponding to the remaining style bits, stylesLeft is set to FE (11111110). Only bits corresponding to requested styles in macStyle are considered. That is, if macStyle is set to 03 (0011) in the above scenario, ATM software will set stylesLeft to 02 (0010).

A value is added to one or more values in the coords parameter. This means that if the input coords values are at the top of the dynamic range of the corresponding axis, newCoords can receive a value outside of the recommended range. This is legal and might look fine for some fonts or for special effects; however, going outside the dynamic range of an axis is generally not recommended.

Note The styles returned in stylesLeft (Outline, Shadow, and so on) can be handled by the ATM software’s usual mimicking of QuickDraw’s style derivation.

convertCoordsToBlendATM

ATMErr convertCoordsToBlendATM(
    Fixed *coords,
    Fixed *weightVector);

convertCoordsToBlendATM( ) converts the user design coordinates for the current font (that is, the one set by TextFont( )) to a weight vector. The returned weight vector is data that can be passed (after conversion from Fixed to Float) to the PostScript language operator makeblendedfont.

coords is an array of Fixed user design coordinate values that are to be converted to a weight vector.

weightVector returns an array of Fixed values that represent the weight vector corresponding to coords for the current font.

Note It is not possible to convert from weightVector coordinates to design coordinates, because there can be more than one set of design coordinates corresponding to a given weight vector.

This function is necessary only for applications that output PostScript language code directly.

See also: makeblendedfont in Appendix A.
**copyFitATM**  

```c
ATMErr copyFitATM(
    short   method,
    Fixed   targetWidth,
    Fixed  *beginDesignCoords,
    Fixed  *baseWidths,
    Fixed  *resultWidth,
    Fixed  *resultDesignCoords);
```

`copyFitATM()` helps perform the final touches of copyfitting with a multiple master typeface. It can be used to adjust the width of a line of text by selecting an instance that obtains a close fit to a specific width.

`method` specifies how the copyfitting will be performed. Currently, ATM software supports three methods:

0  `ATMCopyFitDefault`–let ATM software choose the best method for copyfitting.
1  `ATMCopyFitConstStems`–keep stem widths constant.
2  `ATMCopyFitVarStems`–allow stem widths to vary.

`ATMCopyFitConstStems` tells ATM software to leave the stem widths constant when adjusting the design coords, which might be good for justification. Hence, ATM software tries to keep the letters looking similar between lines.

`ATMCopyFitVarStems` tells ATM software to allow the stem widths to vary, so that the color of the line remains constant, which might be good for headlines (keep the specified weight).

`ATMCopyFitDefault` tells ATM software to choose a method for justifying a line of text. This might change as new algorithms become available. Any other choice will use the default.

`targetWidth` is the final width of the text.

`beginDesignCoords` is the user design coords before copyfitting is to take place.

`baseWidths` is the width(s) of the line of text as if drawn with each of the master designs.

`resultWidth` returns the closest width ATM could achieve to `targetWidth`.

`resultDesignCoords` returns the user design coordinates needed to use with `showTextDesignATM()` to help achieve the target width.
Scenario

First, lay out the line as close as you can get it without adding or subtracting white space. Pass the beginning (current) user design coordinates in `beginDesignCoords`. Using `getMasterFONDATM()` to get the FONDs, get the width of the line of text as if it were drawn using each of the master FONDs, and put the result into the array `baseWidths`. Pass the length you want the final text to be in `targetWidth`.

`copyFitATM()` will return the closest width ATM software can achieve in `resultWidth` and the user design coordinates needed to achieve this width in `resultDesignCoords`. Fill out any discrepancy in `resultWidth` and `targetWidth` by using `spExtra` and/or `chExtra`, and use `resultDesignCoords` with `showTextDesignATM()` to display the results.

The above algorithm will work without ATM software knowing how the line is laid out. However, if some of the characters are manually kerned, the application cannot find the kerning value by looking at the master FONDs. Hence, you'll have to adjust the kern by your own algorithm. For example, you might adjust the manual kern according to the width of the character pair you are kerning.

The following code fragment illustrates the above scenario.

```c
// assume stringWidth is the original string width and that
// targetWidth has been set to the width to be achieved.
// copyfit needs the string length of the string in each of the
// master designs so:
atmErr = getNumMastersATM ( &numMasters ); // find out how many
//...master design
for (i = 1;i <= numMasters; i++) // loop on the master fonds
  // getting the string width for each.
  {
    atmErr = getMasterFONDATM ( i, &masterFOND);
    TextFont(masterFOND);
    TextSize(mmfpReply.size);
    baseWidths[i - 1] = INTTOFIXED (StringWidth (byteString));
  }

atmErr = copyFitATM (ATMCopyFitDefault, targetWidth, coords,
  baseWidths, &resultWidth, mmfpReplyCoords);

// at this point, you may want to compare targetWidth to
// resultWidth and make up the difference (if there is any)
// with spExtra and/or chExtra. Finally, show the text.
atmErr = showTextDesignATM  (   
  NULL, // must be NULL
  (Byte *) &byteString, // The string to show
  strlen(&byteString),
  &m, // the matrix
  mmfpReplyCoords, // the coordinates
  NULL, // No displacements
  &lengthDisplayed);
```
createPermBlendedFontATM  ATMErr createPermBlendedFontATM( 
    StringPtr   fontName,
    short       fontSize,
    short       fontFileID,
    short*      *retFondID );

createPermBlendedFontATM( ) creates a permanent (remains after re-booting), blended font in the file identified by fontFileID.

fontName is the name the font will be given.

fontSize. If nonzero, fontSize will create a bitmap of the current font. However, only the zero case is currently supported, which will create an empty bitmap at size 10 points. (This is a special 'NFNT' that only stores a bitmap for the space character).

fontFileID is the file ID of the resource that the font is to go into.

retFondID returns the font ID that can be used with TextFont( ).

Fonts can be created in any existing font suitcase or resource file using fontFileID. However, the suitcase that the master font is in might be the most natural location. The font ID is returned in retFondID. To ensure that documents using a multiple master font created on another machine will display properly, an application must store the full font name in the document.

Point startPoint;
int numTypes;
SFTypeList typeList;
startPoint.h = 200;
startPoint.v = 250;
umTypes = 1;
typeList[ 0 ] = 'FFIL';
// select a file to put the perm font into, may be better to
// stick it in the suitcase that contains the "master" font.
SFGetFile ( startPoint, "\p", 0L, numTypes,
    &typeList, 0L, replyPtr );
permFondFileID = OpenResFile (reply.fName); // get the ID
atmErr = createPermBlendedFontATM ( 
    (StringPtr) blendName, // from encodeBlendedFontATM
    (short) 0,         // currently must be zero
    permFondFileID, // from OpenResFile()
    &retFONDID );
AppendMenu (gFontMenu, blendName);// add the new font to the menu
// *** do FlushVol, error checking etc.

Note To ensure that the suitcase is not damaged if the system crashes before application termination, the application should perform a FlushVol( ) after creating permanent fonts.

See also: disposePermBlendedFontATM( )
createTempBlendedFontATM function is used to create a temporary blended font. The temporary font can be used for sample text when displaying adjustments made by end users while creating a new blended font instance.

**Note** If you want to show a font with different design axes coordinates and don’t need a temporary FOND, `showTextDesignATM()` will give better performance than creating a temporary instance then showing it.

- **numAxes** is the number of axes in the font to be used.
- **coords** is the design coordinates to be used, they should be in the range of userMin to userMax.
- **useFondID** returns the value of a temporary FOND that should be used to reference the temporary font. Do not refer to the temporary font by name.

**Note** `useFondID` must be initialized to zero for the first call.

Like the ATM software text showing calls, this call references the FOND last set by QuickDraw’s `TextFont()` function, so be sure to select the proper font before calling `createTempBlendedFontATM()`.

Fonts created with this mechanism will not show up in font menus, additionally, the format of the name is different than that of a permanent font; therefore, an application must use `decodeBlendedFontNameATM()` to parse the font name.

Any font created using this mechanism can be drawn, saved to a PICT, and printed just like any other font, if you are using any of the API `ShowText` calls.

**Scenario**

The user wants to create and modify a multiple master font for displaying fonts with new blend coordinate values.

First, set the current font to a multiple master font using `TextFont()`. Then, call `createTempBlendedFontATM()` with `useFondID` initialized to zero, and a new temporary FOND ID is returned in `useFondID`. This temporary FOND ID can be used with `TextFont()` to set the new font. Any of the text showing calls can then be used to display sample text. Typically, more than one
instance is viewed. In this case, the application will loop on the above, supplying `createTempBlendedFontATM()` with the returned FOND ID (initialized to zero only the first time through) and a new set of coordinates.

The `disposeTempBlendedFontATM()` routine is used to eliminate a temporary font. If they are not disposed of, they will be eliminated at the next reboot. To create a permanent font, use `createPermBlendedFontATM()`.

*Note* The application should probably be using `MMFontPickerATM()` or `ShowTextDesignATM()` instead of creating a temporary font.

See also: `showTextDesignATM()`, `disposeTempBlendedFontATM()`, `MMFontPickerATM()`.

### decodeBlendedFontNameATM

<table>
<thead>
<tr>
<th>Function: <code>decodeBlendedFontNameATM()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATMerr</strong> decodeBlendedFontNameATM (</td>
</tr>
<tr>
<td>StringPtr blendName,</td>
</tr>
<tr>
<td>StringPtr familyName,</td>
</tr>
<tr>
<td>short *numAxes,</td>
</tr>
<tr>
<td>Fixed *coords,</td>
</tr>
<tr>
<td>StringPtr displayInstanceStr );</td>
</tr>
</tbody>
</table>

Given a blended font name, returns a family name for the font, the number of axes in the font, the coordinates within those axes, and the display instance string.

*blendName* is a blended font name to be decoded. For example, MyriaMM_500 wt 120 wd.

*familyName* is returned. It is the information before the underscore. For example, MyriaMM in the previous example.

*numAxes* is returned. This is the number of axes in the font. For example, Weight+Width=2.

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

*displayInstanceStr* is returned. This is the value after the underscore. For example, 500 wt 120 wd in the previous example.

Any output parameters can be NULL, in which case this routine ignores them.

*Note* This call can also be used on fonts created using `createTempBlendedFontATM()`, in which case, `displayInstanceStr` is ignored.
Because `decodeBlendedFontNameATM()` parses the `blendName` string and breaks it into sub-strings, it provides a simple method of obtaining a family name from a full PostScript font name.

See also: `encodeBlendedFontNameATM()` for information on converting a multiple master PostScript language font name into a blend name that can be used with the API and `GetFNum()`.

```
disposeTempBlendedFontATM ATMErr disposeTempBlendedFontATM ( short fondID );
```

`disposeTempBlendedFontATM()` is used to eliminate a font created with `createTempBlendedFontATM()` and clean up its tables in ATM memory.

`fondID` is the FOND identification number returned by `createTempBlendedFontATM()`.

See also: `createTempBlendedFontATM()`.

```
disposePermBlendedFontATM ATMErr disposePermBlendedFontATM ( short fondID );
```

`disposePermBlendedFontATM()` is used to dispose of a permanent font and clean up its tables in ATM software memory.

`fondID` is the fond identification number. This is the same as that returned by `createPermBlendedFontATM()` and `GetFNum()`.

See also: `createPermBlendedFontATM()`.

```
codeBlendedFontNameATM ATMErr encodeBlendedFontNameATM ( 
    StringPtr   familyName,
    short       numAxes,
    Fixed       *coords,
    StringPtr   blendName );
```

`encodeBlendedFontNameATM()` when given a family name for a font, the number of axes in the font, and the coordinates within those axes, returns a new name suitable for using with the Macintosh Font Manager and with ATM software API calls.

`familyName` is the family name of a font. It is the information before the underscore in a blended font name. For example, MyriaMM.

`numAxes` is the number of axes in the font. For example, Weight + Width = 2.
coords is an array of Fixed values that represent the font’s coordinates in user space (range userMin – userMax) for each axis.

blendName is returned. This is the blended font instance name that can be used with API calls requiring a font name (also called a blend name) string. An example blend name of a font from the Myriad multiple master font with weight = 450 and width = 600 is: MyriaMM_450 wt 600 wd. Note that the display instance string will not always be delimited by spaces.

The returned blendName can be used with:

getBlendedFontTypeATM( ),
cREATEPermBlendedFontATM( ),
decodeBlendedFontNameATM( ),
MMFontPickerATM( ),
GetFNum( ), etcetera.

See also: decodeBlendedFontNameATM( )

The combination of the decodeBlendedFontNameATM( ) and encodeBlendedFontNameATM( ) procedures can also be used to convert from a PostScript language font name to a blended font name used with API calls and GetFNum( ). For example, from MyriadMM_450 _RG _600 _NO to MyriaMM_450 RG 600 NO.

To achieve this, pass the PostScript language name as the blendName parameter to decodeBlendedFontNameATM( ). Next, pass the output of the decode call into encodeBlendedFontNameATM( ). The resulting blendName will be the correct format to use with the API functions and GetFNum( ).

flushCacheATM ATMErr flushCacheATM ( void );

flushCacheATM( ) flushes the ATM software font cache. This flushes the entire cache, not just information related to the current application; therefore, flushCacheATM( ) should not generally be used by applications. Some applications such as font editors and font handling software might want to flush the cache.
fontFitATM

ATMerr fontFitATM (  
  Fixed *origCoords,  
  short numTargets,  
  short *varyAxes,  
  Fixed *targetMetrics,  
  Fixed **masterMetrics,  
  Fixed *retCoords,  
  Fixed *retWeightVector);

fontFitATM( ) performs constraint matching or “fitting” on the current multiple master typeface. It will allow one or two constraints of arbitrary information or “values.” The values must be information obtainable for each master design of the typeface. Examples of possible constraints are string width, stem width, x height, cap height, color (greyness), stroke width, and so on. Additionally, this function allows the application to choose which axes ATM software can vary to achieve the constraint match.

origCoords is the user design coordinates before the font fitting takes place.

numTargets is the number of constraints. This must be 1 or 2. The number of elements in targetMetrics, masterMetrics, and the number of axes in varyAxes must all be equal to numTargets.

varyAxes are the indices of the axes to vary. Indexed from 1.

targetMetrics is a pointer to the constraint value(s) that are to be achieved.

masterMetrics is a pointer to pointer(s) that contain the original metric values for each master design of the typeface. masterMetrics contains numTargets pointers. Each of these points to n elements, where n is the number of master designs in the typeface. See copyFitATM( ) for an example of looping through master designs to obtain information.

retCoords returns the coordinates that result in the desired constraint matching. If NULL is passed in, retCoords is ignored.

retWeightVector returns the weight vector values. These can be used with the PostScript language makeblendedfont operator and other calculations. If NULL is passed in, retWeightVector is ignored.

Note that the weight vector can be used to calculate information about the given instance. For example, if a value, stemWidth, of the instance is wanted, the dot product of the weight vector and the vector containing the stemWidth for each of the master designs will yield the stemWidth for the instance.

See also: getFontSpecsATM( ), and copyFitATM( ).
**getAxisBlendInfoATM**

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

getAxisBlendInfoATM( ) returns information about the specified axis of the current font.

*axis* identifies the axis (indexed from 1) for which information is being requested.

*userMin* returns the lowest value allowed on this axis.

*userMax* returns the highest value allowed on this axis.

*axisType* returns the type of axis, for example, weight and width. This can be used to determine the meaning of the axis and is constant across all multiple master fonts.

*axisLabel* returns the axis label that can be used in dialogs and so on. Examples of a label are Weight, Width, and so on. This will change across foreign languages.

*axisShortLabel* returns the short version of the axis label to be used in the font name. For example, wt and wd.

**Note**  
See Appendix E for more information on labels.
getBlendedFontTypeATM  short getBlendedFontTypeATM (  
  StringPtr  fontName,  
  short  fondID );

getBlendedFontTypeATM( ) returns whether or not the font specified by  
fondID (or fontName if not NULL) is a blended font usable via ATM software.  
The following are the result codes:

0  ATMNotBlendFont—not a blended font  
1  ATMBlendFontInstance—a blended font instance  
2  ATMBlendFontBaseDesign—a blended font master design  
3  ATMTempBlendFont—temporary blended font instance  

fontName is the name of the font to be checked. This can be NULL, in which  
case fondID will be used to identify the font.

fondID is used to identify the font if the fontName is NULL.

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

getAddressFontNameATM( ) gets the ith primary (also called “blessed” by  
the designer) font name.

i is the index into the “blessed” fonts. The index starts at 1.

blessedFontName returns the blessed font name.

coords returns the user coordinates of the blessed font.

See also: getNumBlessedFontsATM( ) and getRegularBlessedFontATM( )

getAddressFamilyFONDATM  ATMErr getFontFamilyFONDATM (  
  StringPtr  familyName  
  short  *retFondID );

getAddressFamilyFONDATM( ) gets a fond ID for a given family.

familyName is the family name of the fond to be retrieved. For example,  
MyriaMM. familyName can be retrieved using  
decodeBlendedFontNameATM().  

retFondID returns the FOND ID of the first fond of familyName family found.
**getFontSpecsATM**

```c
ATMErr getFontSpecsATM ( FontSpecs *specs );
```

**getFontSpecsATM()** gets the specifications for the current multiple master font. This information can be used with **fontFitATM()**.

*specs* returns a pointer to a *FontSpecs* structure that will be filled in.

As can be seen below, the structure (defined in *ATMinterface.h*) filled in using **getFontSpecsATM()** contains information not available in the FOND. Be careful not to use data from the structure unless the data’s corresponding boolean value is true.

The following is a listing of the *FontSpecs* structure.

```c
typedef struct FontSpecs {
    short version; /* Application must set to zero. */
    Boolean vertStemWidthAvail : 1; /* Signals that data is available */
    Boolean horizStemWidthAvail : 1;
    Boolean xHeightAvail : 1;
    Boolean capHeightAvail : 1;
    Boolean serifWidthAvail : 1;
    Boolean serifHeightAvail : 1;
    long SpecsReserved : 26; /* Must be initialized to zero! */
    Fixed SpecsVertStemWidth; /* Thickness of the vert stems. */
    Fixed SpecsHorizStemWidth; /* Thickness of the horiz stems. */
    Fixed SpecsxHeight; /* The height of the letter 'x'. */
    Fixed SpecsCapHeight; /* The height of a cap letter. */
    Fixed SpecsSerifWidth; /* The width of a serif. */
    Fixed SpecsSerifHeight; /* height of a serif. - how tall */
                        /* are the tips off the base line. */
} FontSpecs;
```

**Note**  *SpecsReserved must be initialized to zero.*

As of ATM software version 3.0, only the *VertStemWidth, xHeight, capHeight* are used. This might change in the future.

See also: **fontFitATM()**
getMasterFONDATM    ATMErr getMasterFONDATM ( short i, short *masterFOND );

    getMasterFONDATM( ) gets the FOND ID for the $i^{th}$ master design 
    ($1 \leq i \leq \text{getNumMastersATM}( )$) for the current multiple master font.

    $i$ is the index that specifies the master design.

    masterFOND returns the FOND ID for the $i^{th}$ master design.

    \textbf{Note} The FOND’s name for the master design might not be in the same format as
    the other multiple master typefaces, so be sure to use the API routine 
    decodeBlendedFontNameATM( ) when parsing it.

    This is intended to be used with \texttt{copyFitATM( )} and \texttt{fontFitATM( )}.

    See also: \texttt{getNumMastersATM( )}.

getNumAxesATM    ATMErr getNumAxesATM ( short *numAxes );

    getNumAxesATM( ) gets the number of design axes for the current multiple 
    master font.

    \texttt{numAxes} returns the number of axes in the current font.

getNumBlessedFontsATM    ATMErr getNumBlessedFontsATM ( short *numBlessedFonts );

    getNumBlessedFontsATM( ) gets the number of instances “blessed” by the 
    font designer.

    \texttt{numBlessedFonts} returns the number of primary font instances in the current 
    font.

    See Also: \texttt{getBlessedFontNameATM( )}, \texttt{getRegularBlessedFontATM( )}.

getNumMastersATM    ATMErr getNumMastersATM ( short *numMasters );

    getNumMastersATM( ) gets the number of master designs in the current font.

    \texttt{numMasters} returns the number of master designs in a multiple master font.

    See Also: \texttt{getMasterFONDATM( )}
getRegularBlessedFontATM  ATMErr getRegularBlessedFontATM ( short *regularIndex );

getRegularBlessedFontATM( ) gets the font index of the instance blessed by the designer as being the regular or normal instance. This can be used by applications to denote a default instance when selecting a multiple master family.

regularIndex returns the index of the instance considered the regular/normal primary font of the family. It is use with getBlessedFontNameATM( ).

See also: getNumBlessedFontsATM( ), getBlessedFontNameATM( ).

getTempBlendedFontFileIDATM  ATMErr getTempBlendedFontFileIDATM( short fileID );

getTempBlendedFontFileIDATM( ) gets the file ID of ATM’s temporary suitcase.

fileID is the file ID of ATM software’s temporary resource file.

MMFontPickerATM  AtmErr MMFontPickerATM ( 
    struct MMFP_Parms *parms,
    struct MMFP_Reply *reply );

MMFontPickerATM( ) can be used to display two different dialogs and perform different functions. As will be shown, which dialog to display is specified by setting one of the MMFP_parms flags.

The default dialog is called the Picker. The Picker allows the user to navigate through the design space of the multiple master font for the selected family name. During this navigation, by dragging the axes sliders, sample text at any given point in the space can be viewed. See Figure 5 for an illustration of the Picker dialog. Note that this dialog can be easily modified by setting MMFP_Parms, which are described below.
The second dialog can be selected by setting the flags variable in `MMFP_parms` to `CREATOR_DIALOG`. The Creator dialog allows the user to navigate through the design space of the multiple master font for the selected family name. During this navigation, by dragging the axes sliders, sample text at any given point in the space can be viewed. The user can then create a new instance of a font, remove an existing instance, list and create all the Primary instances, and print a sample page of a given instance. Figure 6 is an illustration of the Creator dialog.

Figure 5  Picker dialog: default sample text for a Myriad font instance.

![Picker dialog](image)

Figure 6  Creator dialog: default sample text for a Myriad font instance.

![Creator dialog](image)
Multiple Master Font Picker Input Parameters – MMFP_parms

The following is a list of the multiple master font picker input parameters – MMFP_parms.

>Note All strings are Pascal format.

```c
MMFP_Parms
struct MMFP_Parms { /* input parameters */
    short version; /* always 1 */
    short flags; /* option flags, 0 for default Picker dialog */
    Point where; /* dialog's top-left corner, [0,0] for default */
    char *prompt; /* prompt string, NULL for default */
    char *sample; /* sample string, NULL for default */
    short startFondID; /* initial font, -1 for none */
    char *startFamilyName; /* initial family, NULL for default */
    Fixed *startCoords; /* initial axis coordinates, NULL for default */
    short startSize; /* initial sample size, 0 for default */
    MMFPHook dlgHook; /* application dialog hook fn, NULL for none */
};
typedef struct MMFP_Parms MMFP_Parms, *MMFP_ParmsPtr;
```

The MMFP_Parms apply to both the Font Picker and Font Creator dialogs. Set the CREATOR_DIALOG bit in flags to get the Creator rather than the Picker dialog.

**where** specifies the dialog's position, or use [0,0] to get the default positioning, which is the center of the main screen.

**prompt** is a “prompt string” that can be passed in. Use NULL to get the default prompt, which is “Create Or Remove any font” for the Creator dialog and “Choose A Font” for the Picker dialog.

**sample** is a sample string that can be passed in. Use NULL to get the default sample, which is the string “Sample.” The sample string is continuously redrawn to illustrate the current font as the user navigates through the multiple master design space. A short fragment of the user’s selection in the current document is a reasonable alternative to the default sample text. The user is also able to type in the sample box to see other characters.

**startSize** is the initial size of the sample text. Use 0 to get the default initial sample size which is 36 points. The user can also adjust the sample size through a popup menu and type-in size box.

**dlgHook** is a hook for a dialog event hook function that can be supplied by an application if a change in the dialog behavior or appearance is wanted. If a dlgHook is provided, it is called by the Picker immediately after each call to ModalDialog(). Refer to the “Standard File Package” chapter of Inside Macintosh for more details on how to write and use a dialog hook routine.
**startFondID, startFamilyName, startCoords:** When the Picker or Creator dialog is first displayed, the family and instance popup menus and the sliders and axis values are all set to reflect the initial font. Additionally, the sample string is rendered in this initial font at the initial start size.

There are a number of ways to choose this initial font. If `startFondID` is a multiple master font then it is used as the initial font. If `startFondID` is –1 or a non-multiple master font, then `startFamilyName` is checked. If this is the name of a multiple master font, then the instance at `startCoords` is used as the initial font.

`startCoords` must have as many elements as there are design axes for the specified family. If `startCoords` is `NULL`, then the first instance in `startFamilyName` is used as the initial font. If `startFamilyName` is `NULL`, then `startCoords` is ignored and the regular style font is chosen.

**Multiple Master Font Picker Reply Parameters: MMFP_Reply**

While the user manipulates the Picker or Creator dialog's controls to navigate through the multiple master design space, the sample text is continuously rendered in the font instance with the selected design space coordinates. A temporary instance is quickly created whenever a permanent instance does not already exist. Only the permanent instances remain when the dialog is dismissed.

The Creator dialog has only an OK button while the Picker dialog has both OK and Cancel. (For both dialogs, OK can be re-titled Quit using the `OK_IS_QUIT` flag bit.) The Picker call returns `noErr` (0) for OK, and –1 for Cancel. (This might change to a more specific ATM software error code in the future.) Regardless of the way the user dismisses the dialog, any fields specified in `MMFP_Reply` are filled in to reflect the state in which the user last left the dialog.

**Note** All strings are Pascal format. Also, be aware that any pointer argument specified as `NULL` is ignored; all others are filled in.
struct MMFP_Reply
{
    char* sample;  /* last sample string (Str255), NULL to ignore */
    short fondID;  /* selected font if permanent, else -1 */
    char *familyName;  /* selected family (Str32), NULL to ignore */
    short numAxes;  /* number of design axes in selected family */
    Fixed *coords;  /* coords of selected instance, NULL to ignore */
    short size;  /* last sample size */
};

typedef struct MMFP_Reply  MMFP_Reply, *MMFP_ReplyPtr;

*sample* is specified to get a copy of the user's sample text string.

*fondID* is the font ID corresponding to the permanent instance, if one exists. If the instance was temporary, it no longer exists, so *fondID* is set to –1.

*familyName* is specified to get the name of the current multiple master family. This name is returned as an ATM software 5-3-3 name, that is, MyriaMM_500 wt 120 wd.

*numAxes* is the number of axes the select font contains.

*coords* is specified to get an array of user design coordinates for the current instance. *coords* will contain *numAxes* elements. Use *MaxBlendAxes* to safely allocate an array long enough for the returned coordinates. If the font instance corresponding to these design coordinates was a permanent one, its ID is returned. If the instance was a temporary one, it no longer exists, so *fondID* is set to –1.

*size* returns the sample size last chosen by the user.

The creator places newly created instances in the same suitcase in which the master fonts are found.

If the “picker” is used, the calling program can make appropriate ATM software calls to create a new temporary or permanent instance of that font by using the *familyName* and *coords*.

Despite their different appearances, both the Creator and Picker dialogs share the same item numbering. The locations of items might change, but the numbering will remain the same.
normToUserCoordsATM  ATMErr normToUserCoordsATM(
    Fixed  *normalCoords,
    Fixed  *coords );

normToUserCoordsATM( ) converts normalized design coordinates for the current font to user design coordinates.

normalCoords is an array of normalized coordinates to be converted to user coordinates.

coods returns the user coordinates as an array of Fixed values.

Note  All the API calls that deal with coordinates require user coordinates (except userToNormCoordsATM( ) and normToUserCoordsATM( )). Normalized coordinates are provided because they are convenient to use for mathematical manipulations.

See also: userToNormCoordsATM( ).

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

showTextDesignATM( ) extends the previous ATM software ShowText calls so that ATM software can display strings without needing an explicit FOND. It is convenient for displaying temporary instances of multiple master fonts.

fontFamily specifies the PostScript language font you want to display. If NULL, ATM software uses the fond ID specified in the current GrafPort.

Note  In version 3.0 of ATM, only the NULL case of fontFamily is supported.

text is a pointer to the text string to be drawn.

len is the number of characters in the string to be drawn.

matrix is the matrix that describes the transformation to be applied to the string when rendered. For more information, see the section “Drawing Text With The ATM API” in Technical Note #5072, Adobe Type Manager Software API: Macintosh.
coords is an optional parameter that specifies the user design coordinates of a multiple master typeface without the need to create a new FOND. Pass a NULL if the font is not a multiple master typeface, or if fontFamily is NULL and you want to use the design coordinates unchanged from the current FOND.

displacements is a pointer to an array of fixed point numbers. The number of the elements in the array is two times the number of characters of the text string being imaged. This displacement array is a series of x and y widths for the characters. It is possible to have different widths for a given character that appears more than once in the string. This allows for both track kerning and pairwise kerning, as well as special affects.

If displacements is NULL, either the default FOND widths or the standard character widths of the PostScript file fontFamily (if fontFamily is not NULL) are used.

For more information on the displacement array, see the section “Drawing Text With The ATM API” in Technical Note #5072, “Adobe Type Manager Software API: Macintosh”.

lenDisplayed returns the number of characters drawn. This is helpful in error recovery, that is, QuickDraw can be used to image the characters not displayed).

Note Unlike showTextDesignATM( ), the other ATM text showing calls such as xyShowTextATM( ) return the number of characters not rendered.

Whenever possible, showTextDesignATM( ) should be used instead of a combination of createTempBlendedFontATM( ) and one of the other showText calls. showTextDesignATM( ) is faster and requires fewer resources.
// have the user pick a font to preview
atmErr = MMFontPickerATM (&mmfpParams, &mmfpReply);
// Set the font to be used by showTextDesignATM
atmErr = getFontFamilyFONDATM ((StringPtr) mmfpReply.familyName, &useFondID);
TextFont(useFondID);
// show the font
atmErr = showTextDesignATM {
   NULL,                  // family stuff not implemented
   &byteString,           // the string to show
   strlen(&byteString),   // the transformation matrix
   &m,
   mmfpReply.coords,      // the design coords from picker
   NULL,                  // no displacements
   &lengthDisplayed );
// ...now do some error checking

userToNormCoordsATM
ATMErr userToNormCoordsATM ( 
   Fixed  *coords, 
   Fixed  *normalCoords );

userToNormCoordsATM( ) converts the user design coordinates for the current font to normalized design coordinates.

coords is the user design coordinates to be converted to normalized coordinates.

normalCoords returns an array of Fixed normalized coordinates.

Note All API calls that deal with coordinates (except userToNormCoordsATM( ) and normToUserCoordsATM( )), require user coordinates. Normalized coordinates are provided because they are convenient to use for mathematical manipulations.

See also: normToUserCoordsATM( ).
Appendix A: The makeblendedfont Operator

blendedFontDict weightVector makeblendedfont – blendedFontDict

makeblendedfont creates a font dictionary with blended entries. The blendedFontDict is a font dictionary for a blended font, and weightVector is an array of numbers summing to 1.0 to be used as the weights for creating the new blended font.

The result is a complete font dictionary that can be used as an argument to the definefont operator. The resultant dictionary blendedFontDict and its contents are still read-write, so the caller of makeblendedfont can make further modifications if necessary (such as assigning a UniqueID). The makeblendedfont operator will not copy FID’s, XUID’s, or UniqueID’s.

The makeblendedfont operator will become a standard feature of the PostScript language and will be defined in systemdict as part of the interpreter. For backward compatibility, the downloadable text file for a multiple master typeface should include conditional code that will define makeblendedfont if not implemented in the interpreter.

For more information on multiple master fonts and the makeblendedfont operator, see Adobe Technical Specification, “Adobe Type 1 Font Format: Multiple Master Extensions”.
Appendix B: 5-3-3 Rule for Macintosh File Names

The Macintosh environment has a file naming scheme that is constructed from the PostScript font name. It is called the 5-3-3 rule. A file name is constructed by dissecting the PostScript font name into components based on capital letters and hyphens. The first five letters of the first name component are used, followed by the first three letters of any subsequent name components. The following examples illustrates this concept.

- **Palatino-Roman** => **PalatRom**
- **Palatino-Bold-Italic** => **PalatBolIta**
- **Optima** => **Optim**
- **Optima-Bold-Oblique** => **OptimBolObl**

Notice in the following example that dissections can occur at capital letters and that it is not necessary to fulfill a complete string of five or three characters (for example, the capital A in the following name).

- **AGaramond-BoldItalic** => **AGarBolIta**

```
 Palatino-Roman   =>   PalatRom
 Palatino-Bold-Italic => PalatBolIta
 Optima           =>   Optim
 Optima-Bold-Oblique => OptimBolObl
```

```
AGaramond-BoldItalic   =>   AGarBolIta
5 3 3 3 3
```

```
MyriadMM_500_wt_600_wd => MyriaMM, all instances are from the same font file.
```

```
MinionMM-Italic_367_RG_365_CN_11_OP => MinioMMIta
```
Appendix C: ATM software
Error Codes

The following are ATM software error codes as defined in `ATM.h`.

```c
#define ATM_NOERR (0) /* Normal return */
#define ATM_NO_VALID_FONT (-1) /* can't find proper outline font*/
#define ATM_CANTHAPPEN (-2) /* Internal ATM error */
#define ATM_BAD_MATRIX (-3) /* Matrix undefined or matrix too big*/
#define ATM_MEMORY (-4) /* Ran out of memory */
#define ATM_WRONG_VERSION (-5) /* currently installed ATM driver doesn't support this interface */
#define ATM_NOT_ON (-6) /* the ATM driver is missing or has been turned off */
#define ATM_FILL_ORDER (-7) /* inconsistent fill calls, e.g. () without ATMStartFill() */
#define ATM_CANCELLED (-8) /* the client halted an operation, e.g. a callback from ATMGetOutline returned 0 */
#define ATM_NO_CHAR (-9) /* the font does not have an outline for this character code */
#define ATM_BAD_LENGTH (-10) /* ATMShowText() or ATMxyShowText() was called with length argument <= 0 or > 255 */
#define ATM_NOT_BLENDED_FONT(-12) /* This font is not a blended font. */
#define ATM_BASEDESIGN (-13) /* This operation is not allowed on a base design (eg. deleting FOND).*/
#define ATM_TEMPPFONT_PROB (-14) /* ATM had a problem working with a temporary font. */
#define ATM_ILL_OPER (-15) /* Can't perform this operation on this font. */
#define ATM_FONTFIT_FAIL (-16) /* FontFit() failed (also from CopyFit()!). */
#define ATM_MISS_BASEDESIGN (-17) /* Missing base design FOND. */
```

/* (-11) Not in use in procs version 5*/
#define ATM_NOBLENDED_FONTS (-18) /* no multiple master fonts installed*/

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

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

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

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

#define ATM_IO_ERROR (-23) /* I/O error*/
Appendix D: Function Prototype Summary

**addMacStyleToCoordsATM**

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

**convertCoordsToBlendATM**

```c
ATMErr convertCoordsToBlendATM ( 
    Fixed  *coords,
    Fixed  *weightVector );
```

**copyFitATM**

```c
ATMErr copyFitATM ( 
    short  method,
    Fixed  targetWidth,
    Fixed  *beginDesign,
    Fixed  *baseWidths,
    Fixed  *resultWidth,
    Fixed  *resultDesign );
```

**createPermBlendedFontATM**

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

**createTempBlendedFontATM**

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

ATMErr decodeBlendedFontNameATM ( 
  StringPtr blendName,
  StringPtr familyName,
  short *numAxes,
  Fixed *coords,
  StringPtr displayInstanceStr );

disposeTempBlendedFontATM

ATMErr disposeTempBlendedFontATM ( 
  short FONDID );

disposePermBlendedFontATM

ATMErr disposePermBlendedFontATM ( 
  short fondID );

encodeBlendedFontNameATMATMErr

encodeBlendedFontNameATM( 
  StringPtr familyName,
  short numAxes,
  Fixed *coords,
  StringPtr blendName);

flushCacheATM

ATMErr flushCacheATM ( void );

fontFitATM

ATMErr fontFitATM ( 
  Fixed *origCoords,
  short numTargets,
  short *varyAxes,
  Fixed *targetMetrics,
  Fixed **masterMetrics,
  Fixed *retCoords,
  Fixed *retWeightVector );

getAxisBlendInfoATM

ATMErr getAxisBlendInfoATM ( 
  short axis,
  short *userMin,
  short *userMax,
  StringPtr axisType,
  StringPtr axisLabel,
  StringPtr axisShortLabel );
getBlendedFontTypeATM
short getBlendedFontTypeATM (StringPtr fontName, short fondID);

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

g.getFontFamilyFONDATM
ATMErr getFontFamilyFONDATM (StringPtr familyName, short *retFondID);

g.getFontSpecsATM
ATMErr getFontSpecsATM (FontSpecs *specs);

g.getMasterFONDATM
ATMErr getMasterFONDATM (short i, short *masterFOND);

g.getNumAxesATM
ATMErr getNumAxesATM (short *numAxes);

g.getNumBlessedFontsATM
ATMErr getNumBlessedFontsATM (short *numBlessedFonts);

g.getNumMastersATM
ATMErr getNumMastersATM (short *numMasters);

g.getRegularBlessedFontATM
ATMErr getRegularBlessedFontATM (short *regularIndex);
getTempBlendedFontFileIDATM

ATMErr getTempBlendedFontFileIDATM(
    short fileID);

MMFontPickerATM

AtmErr MMFontPickerATM (  
    struct MMFP_Parms *parms,
    struct MMFP_Reply *reply);

normToUserCoordsATM

ATMErr normToUserCoordsATM(
    Fixed    *normalCoords,
    Fixed    *coords);

showTextDesignATM

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

userToNormCoordsATM

ATMErr userToNormCoordsATM(
    Fixed    *coords,
    Fixed    *normalCoords);
Appendix E: Font Axis and Label Names

To distinguish primary fonts from fonts created by the user, the primary fonts use an uppercase abbreviation of the location within the dynamic range (for example, XL for ExtraLight) while user-created fonts use generic lowercase abbreviations (for example, wt for weight and wd for width). Adobe’s naming convention allows the following uppercase abbreviations for primary font names:

Table 1  Uppercase abbreviations for primary font names

<table>
<thead>
<tr>
<th>Axis type</th>
<th>Generic</th>
<th>Primary</th>
<th>Axis label</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>(wt)</td>
<td>XL</td>
<td>ExtraLight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LT</td>
<td>Light</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RG</td>
<td>Regular</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SB</td>
<td>Semibold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BD</td>
<td>Bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BL</td>
<td>Black</td>
</tr>
<tr>
<td>width</td>
<td>(wd)</td>
<td>XC</td>
<td>ExtraCondensed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN</td>
<td>Condensed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SC</td>
<td>SemiCondensed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SE</td>
<td>SemiExtended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
<td>Extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XE</td>
<td>ExtraExtended</td>
</tr>
<tr>
<td>optical size</td>
<td>(op)</td>
<td>OP</td>
<td>Optical size</td>
</tr>
</tbody>
</table>

See getAxisBlendInfoATM for information about obtaining this information from a font.
Appendix F: Supporting EPS Files

This section discusses topics specific to EPS files and multiple master fonts. For general information on EPS files, see Appendix G of the PostScript Language Reference Manual, Second Edition.

Generating EPS Files

When creating an EPS file, list only the PostScript language family name (one time for each family of fonts used in the document) in the Document Structuring Conventions (DSC) comments such as %%DocumentFonts:. Use the full PostScript language font name, including the instance string in the PostScript language code which invokes the font.

The PostScript language family name can be obtained by passing the full PostScript language font name to decodeBlendedFontNameATM().

The DSC comments might look something like the following:

```plaintext
%%DocumentFonts: MyriadMM
%%+ MyriadMM-Italic

%%DocumentNeededFonts: MyriadMM
%%+ MyriadMM-Italic
```

Output the PostScript language code to reference each multiple master instance using the full PostScript font name. The full PostScript language font name which includes the instance string, can be obtained via the FOND’s Style Mapping table in the usual way (as with non-multiple master fonts).

**Note** There must be a FOND for the font instance for this method to work. Therefore, the application should use createPermBlendedFont() when creating each font instance.
The code will look something like the following:

```plaintext
•••
/sf {scalefont setfont} bind def
•••
/f1 /MyriadMM_450_RG_600_NO findfont def
/f2 /MyriadMM-Italic_570_wt_510_wd findfont def
•••
100 100 moveto
f1 20 sf (this is MyriadMM ) show
f2 30 sf (this is MyriadMM-Italic ) show
f1 50 sf (this is larger MyriadMM ) show
•••
```

Note that the `findfont` operator is redefined in the multiple master font. This re-definition provides for the creation of the multiple master instance if it doesn’t already exist on the printer. This does require that the master font (such as MyriadMM in the example above) exist on the printer.

**Importing EPS Files**

As a primer for this section, remember that the DSC comments of an EPS file can contain any of the following PostScript language font name formats which will be discussed below:

*MyriadMM*

multiple master PostScript language *family* name. As described in the section “Generating EPS Files”, this is what a multiple master aware EPSF generator should produce.

*MyriadMM-Italic* _400_RG_200_CN

Full multiple master PostScript language name including the instance string. This is what a non-multiple master aware EPSF generator will produce.

*Times-Roman*

Non-multiple master font name.

An application importing and printing an EPS file may need to download the fonts that are used in the EPS file. To do this, the FONDs corresponding to the PostScript fonts may need to be located.

**Step 1: Finding The FONDs**

The application can start by looping through the FONDs looking at the style mapping tables for each PostScript language font name found in the DSC comments section of the EPS file. If a match is found (the font already exists in this system), the FOND that contained the match can be used to download
the font. If the font doesn’t need to be downloaded yet, the font name, the Macintosh style, and the FOND ID could be stored in a table for future handling.

If a match isn’t made, the PostScript language name should be kept in a “fonts not yet found” list for further checking. There are three cases that will cause an entry in the list:

- A non-multiple master font not available on the system, or a multiple master font whose master font is not on the system. There is no resolution to this.

- A multiple master instance not available on the system. Since the master font is what gets downloaded to the printer, this can be resolved by finding any FOND in the family and using it to download the font.

- A multiple master family name whose regular instance is not available on the system. Again, this can be resolved by finding any FOND in the family.

The family name of a multiple master font maps to the regular instance of the font. For example, “MyriadMM” will be matched in the Style Mapping Table of the FOND “MyriaMM_400 RG 600 NO” which is the FOND name for the regular instance of Myriad. Therefore, if the regular instance is present on the system, step 1 above will provide a match for the family name.

**Step 2: Resolving problems 2 and 3.**

To check whether a name is a multiple master instance name or a multiple master family name, do a decodeBlendedFontNameATM( ) on it and check the return value. If the name is an instance, ATM_NOERR will be returned and an appropriate FOND can be obtained by passing the family name returned by decodeBlendedFontNameATM( ) to getFontFamilyFONDATM( ).

If an error was returned by decodeBlendedFontNameATM( ), the name may be a family name or a non-multiple master font name. As mentioned in item three above, the family name case can be resolved.

getFontFamilyFONDATM( ) will take a multiple master family name and will return an error on non-multiple master family names. So, pass the name to getFontFamilyFONDATM( ). If the name is a multiple master family name (e.g. “MyriadMM”), ATM_NOERR will be returned. If no error was returned, then the FOND ID returned by getFontFamilyFONDATM( ) can be used to download the font.

The following listing illustrates “Step 2” above.
/* fontName is a string that contains the PostScript */
/* name from the EPS file */
err = decodeBlendedFontNameATM(
    fontName,&familyName,&numAxis,
    &coords,&instanceStr);
if (err){
    err = getFontFamilyFONDATM(fontName,&fondID);
    if (!err)
        // make entry in a table or use it to download
        // the font etc.
}
else{
    err = getFontFamilyFONDATM(familyName,&fondID);
    if (!err)
        // make entry in a table or use it to download the
        // font etc.
}

**Downloading the Font**

The usual method of downloading a font is to image a space character. In some fonts, this may be a visible character so steps should be taken to be sure the character isn’t imaged on the page. One method might be to image outside the printed page. Additionally, be aware that the QuickDraw pen location will change when imaging the character. This will need to be considered by some applications.

**DSC Comments**

Listed below are the three DSC comments that the EPS file reader should be concerned with when determining which fonts to download.

**%%DocumentFonts:**
This lists all the fonts that are *used* in the document.

**%%DocumentSuppliedFonts:**
Lists all the fonts that are *included* within the EPS file.

**%%DocumentNeededFonts:**
Lists the fonts that are used but *not* included in the EPS file.

Unfortunately, not all applications produce proper DSC comments. Here is a strategy an application could use when determining which fonts to download.

- Download only the fonts listed in the **%%DocumentNeededFonts:** section if it exists.
• Download only the fonts that make up the difference in the 
  %%DocumentFonts: list and the %%DocumentSuppliedFonts: list if they 
  both exist.

• Download all of the fonts listed in the %%DocumentFonts: list if it is the 
  only font comment in the file.
Index

A
addMacStyleToCoordsATM 13, 41
AddResMenu 11
Adobe Type Manager software See
ATM software
API Function
addMacStyleToCoordsATM 13
convertCoordsToBlendATM 14
copyFitATM 15
createPermBlendedFontATM 17
createTempBlendedFontATM 18
decodeBlendedFontNameATM
19
disposePermBlendedFontATM
20
disposeTempBlendedFontATM
20
encodeBlendedFontNameATM
20
flushCacheATM 21
fontFitATM 22
getAxisBlendInfoATM 23
getBlendedFontTypeATM 24
getBlessedFontNameATM 24
getFontFamilyFondATM 24
getFontSpecsATM 25
getMasterFONDATM 26
getNumAxesATM 26
getNumBlessedFontsATM 26
getNumMastersATM 26
getRegularBlessedFontATM 27
getTempBlendedFontFileIDATM
27
MMFontPickerATM 27
normToUserCoordsATM 32
showTextDesignATM 32
userToNormCoordsATM 34
ATM software
API routines 13–34
error codes
ATM.h 39
ATM software API
Multiple Master fonts
Macintosh 5–34
ATM.h
error codes 39
ATMCopyFitConstStems 15
ATMCopyFitDefault 15
ATMCopyFitVarStems 15
B
blended font 7
C
convertCoordsToBlendATM 14, 41
copyFitATM 15, 41
createPermBlendedFontATM 17, 41
createTempBlendedFontATM 17, 41
Custom fonts 8
D
decodeBlendedFontNameATM 19,
42
disposePermBlendedFontATM 20,
42
disposeTempBlendedFontATM 20,
42
Document
Carry font names in 10
E
encodeBlendedFontNameATM 20
blendName 21
encodeBlendedFontNameATMATM
Err 42
EPSF
  Downloading fonts 50
  DSC comments in 47
  Generating EPS Files 47
  Importing EPS Files 48
  Sample code for finding FOND IDs 50
  sample PostScript language output 48

F
  flushCacheATM 21
  FOND Names 9
  font axis 45
  Font Names 8
  fontFitATM 22, 42
  FontSpecs structure 25
  function prototype summary 41

G, H
  getAxisBlendInfoATM 42, 45
  getBlendedFontTypeATM 24, 43
  getBlessedFontNameATM 24, 43
  GetFNum patch to 10
  getFontFamilyFondATM 24, 43
  getFontSpecsATM 25, 43
  getMasterFONDATM 26, 43
  getNumAxesATM 26, 43
  getNumBlessedFontsATM 26, 43
  getNumMastersATM 26, 43
  getRegularBlessedFontATM 27, 43
  getTempBlendedFontFileIDATM 27, 44

I, J, K
  instance 7
  interior designs 7

L
  label names 45

M
  Macintosh
    file names
  5-3-3 rule. 37
  macStyle 13
  master design 7
  MMFontPickerATM 27, 44
  MMFP_Parms 29
  MMFP_Reply 31
  Multiple Master fonts
    ATM software API
      Macintosh 5–34
    blendedFontDict 35
    makeblendedfont 35
    picker input parameters 29
    picker reply parameters 30
    scenario using blend coordinate values 18
    weightVector 35
  Multiple Master typefaces
    design axes and dynamic ranges 5
    design space examples 6–7

N, O
  Normalized design coordinates 8
  normToUserCoordsATM 32, 44

P, Q
  Patch to GetFNum() 10
  Picker dialog 28
  PostScript Font Names 10
  primary font names
    uppercase abbreviations 45
  Primary fonts 8

R
  Regular instance 8

S, T
  showTextDesignATM 18, 32, 44
  StringPtr 42
  Suitcase ATM's temporary 11

U, V
  User design coordinates 8
  userMax 8