

New Technical Notes

Macintosh



Developer Support

Fond of FONDS

Text

M.TE.FONDS

Written by: Joseph Maurer

May 1992

This Technical Note takes the place of Tech Note #26, “Character vs. String Operations in QuickDraw” by Bryan Stearns (March 1988), which pointed out the possible differences between the results of a `StringWidth` call and successive calls to `CharWidth`. This Note updates and brings into a broader context the issues related to text measuring. It also provides additional documentation on font family resources ('FOND's), and addresses various other frequently asked questions related to the Font Manager.

Introduction

Every Macintosh developer needs to draw text in a GrafPort, and to specify typeface, size, and style. In most cases, there are no problems, and application developers don't need to have in-depth knowledge of the Font Manager's inner workings and the data structures involved. Sometimes, however, the results on the screen or on printed output may be different from what you expected. Then, usually, DTS comes into play to figure out what the problem is and how to fix it. This Note is based on sharp developer questions from the last year or so, which point mainly at shortcomings of the existing Font Manager architecture, inconsistencies in its data structures, and missing details in the documentation.

We'll start with a historical overview, which discusses the introduction of font family description resources ('FOND's) back in 1986, explains the consequences of non-proportionally scaling fonts, and covers non-registration and volatility of font family numbers.

We will then deal with the Font/DA Mover and the built-in “Mover” of the Finder in System 7. We discuss a number of not-so-well-known aspects of moving fonts in and out of a suitcase file, and recommend that you altogether abandon the resource type 'FONT'. We'll also comment on font names, and show you how to put separate stylistic variants of a typeface together into one font family. And we provide documentation on the `ffVersion` field of a 'FOND' (accompanied by a disclaimer and another piece of irritating information).

The main body of this Note addresses how the Font Manager works in the `FMSwapFont` context, and gives information on the scaling factors in the `FMOutput` structure and on the

changes introduced by TrueType. We again took the examples of unexpected behavior (under certain circumstances) from developer questions. Thanks for helping document this!

Determining the width of text, as required for line layout, is sometimes trickier than you might think. We will document the effects of `SetFractEnable` in more detail and mention some more line layout problems.

Finally, this Note includes sample code that puts the `OutlineMetrics` call to work, and determines text bounding boxes for bitmap fonts.

Some FOND Background

Originally (*Inside Macintosh* Volume I, Chapter 7), all font-related data was contained in resources of type 'FONT'. For a font number within the range 0...255, and a font size restricted to less than 128, the (unnamed) 'FONT' resource with an ID:

$$128 * (\text{font number}) + (\text{font size})$$

contained the bitmap font strike, while the 'FONT' resource with ID = $128 * (\text{font number})$, corresponding to font size 0, did not contain any data, but its resource name provided the font family name. QuickDraw took care of stylistic variants like italic, bold, shadow, and so on; if a user had a specifically fine-tuned font strike for a stylistic variant, QuickDraw would **not** automatically substitute it when drawing text.

For aesthetic reasons, bitmap fonts for different sizes were usually designed with widths non-proportional to the point size. For example, the text “*Show the difference in text widths*” drawn with Courier 9 measures 170 pixels, whereas the same text drawn with Courier 18 measures 374 pixels, which is 10% more than you expect. (By the way, this is bad news for the ImageWriter printer driver. When “Best” mode (144 dpi) is selected and text in Courier 9 is to be printed, the printer driver uses Courier 18 to render the 9-point font size on the paper at twice the screen resolution, and obviously has big trouble compensating for the 10% difference in text width.)

On the other hand, given that only integer character widths (in QuickDraw’s 72 dpi units) are possible, proportional font scaling is compromised anyway. Accumulated rounding errors in text measuring, particularly for scaled fonts, contribute to the headaches of many Macintosh programmers. The computed text widths (vital for positioning text precisely and for line layout algorithms to justify text) sometimes change quite abruptly when the user removes or adds certain font sizes.

The introduction of the LaserWriter, and the success of Macintosh in the desktop publishing arena, required an extension of the original Font Manager architecture. This extension is based on the concept of “font family description” resources of type 'FOND', and on a new resource type 'NFNT' for the data of the existing 'FONT' resources (see *Inside Macintosh* Volume IV, Chapter 5).

The 'FOND' resource stores size-independent information about the font family, and its resource ID is the font number (in the range 0...32767). The resource name of the 'FOND' is the font name, and it contains a variable-length **font association table**, which references the font strikes belonging to a specific font family. These references include size, style, and resource ID of the 'NFNT' or 'FONT' resource containing the bitmap font data. TrueType fonts were retrofitted into this scheme, and are identified as font strike resources for point size zero. Any reference to point size zero refers to a resource of type 'sfnt'.

Note: The range 0...32767 for font numbers is subdivided into ranges for the various script systems (see *Inside Macintosh* Volume VI, pages 13-8 and 14-22, and M.TE.FontsAndScripts). This restricts the range of font numbers for the Roman script to 0...16383, with 0, 1, and 16383 reserved for the system.

Since Apple originally intended fonts to be referenced by their font family numbers, DTS attempted to register those numbers (see *Inside Macintosh* Volume I, page 219 and Volume IV, page 31). This failed—not only because

the number of fonts registered grew greater than the number of font family numbers available, but also because the Font/DA Mover (version 3.8, shipped with System 6), and the “Mover” built into the System 7 Finder resolve conflicts between font IDs (which happened anyway!) by renumbering the fonts on-the-fly. There is no font ID registration any more—except for the very special case of Japanese Kanji 'FOND'-'fbit' IDs, and potentially for Korean, Chinese and other double-byte fonts.

As early as April 1988, M.IM.FontNames recommended the use of font names rather than font family numbers. Since then, the recommendation has been reinforced in *Inside Macintosh* Volume VI, page 12-16. Fortunately, most applications have been good about following this recommendation. Unfortunately, some exceptions remain, even in Apple’s own software. QuickDraw Pictures created without 32-Bit QuickDraw refer to fonts by font family number only!

For obvious reasons of upward compatibility (to maintain existing fonts, and to avoid reflowing of existing documents), the introduction of 'FOND's did not solve all the problems. This is what this Note is all about.

Moofing Fonts

The Font/DA Mover utility has evolved into version 4.1, which knows about 'sfnt's. It is available on the *Developer CD Series* disc, path “Tools & Apps (Moof!): Misc. Utilities:”. The Finder in System 7 incorporates its own “Mover” (see *Inside Macintosh* Volume VI, page 9-33), which makes the Font/DA Mover redundant for System 7 users.

Given the combinatorial explosion of all imaginable situations with 'FOND's, 'FONT's, 'NFNT's and 'sfnt's, and stylistic variations of fonts belonging to the same family, the font moving job deserves respect. The following notes cover some less well-known aspects of this business.

- If an old “standalone” 'FONT' (without corresponding 'FOND' resource) is moved into a suitcase file, Font/DA Mover or the System 7 Mover creates a minimal 'FOND' resource on-the-fly. This 'FOND' has no tables, and nearly all its fields are zeroed. The System 7 Finder also converts the resource type from 'FONT' to 'NFNT'; unfortunately, the Font/DA Mover keeps the resource type 'FONT'.

Note: While it is perfectly legal to have 'FOND's continue to reference the older 'FONT' type, DTS recommends that you avoid 'FONT's. Accessing 'FONT's is much slower, since the Font Manager always looks for 'FOND's and 'NFNT's first. More importantly, 'FONT's are troublemakers if an application comes with its own font in its resource fork.

Imagine an application that includes a private 'FOND' which references a 'FONT' in its resource fork by resource ID. When the Font Manager wants to load the font resource, it first looks for a resource of type 'NFNT' with this same resource ID. If there's an 'NFNT' in the System file with the same resource ID, the Font Manager will pick it instead of the 'FONT' from the application's resource fork. This happens more often than you'd like to think!

- Under the current font architecture, the font name is the resource name of the 'FOND' resource (let's forget about 'FONT's altogether), so the font name can be any Pascal string. Unfortunately, this conflicts with the 31-character limitation of a file name when the System 7 Finder derives the file name of a movable font file (*Inside Macintosh* Volume VI, page 9-34) from the font name. Some third-party fonts come with font names long enough to cause trouble. You may also see this problem when trying to open a suitcase if the Finder can't generate distinct names for all of the fonts in the suitcase; the Finder may say the suitcase is “damaged” when it is not.

Note: Each TrueType 'sfnt' resource contains a Naming Table (see *The TrueType™ Font Format Specification*, APDA™ M0825LL/A) which provides nearly unrestricted font naming capabilities, to accommodate the needs of font manufacturers. A forthcoming Macintosh Technical Note on TrueType Naming Tables gives additional information.

- QuickDraw and the current Font Manager have no provision for stylistic variants like “light,” “medium,” “demi,” “book,” “black,” “heavy,” “extra,” “ultra,” etc., used in the context of professional typesetting. Therefore, each of these variants comes with a separate font family resource. Probably for reasons of consistency, the “italic” variants have their own font family resources as well. Unfortunately, unless each 'FOND' references both the “plain” and the “italic” font strikes, QuickDraw will no longer know a customized italic font strike exists.

It is fairly easy, using System 7 and ResEdit, to merge two font families (named, for example, “myFont” and “myFont italic”) into one. This way, QuickDraw will automatically use the pre-designed italic font strike instead of creating one algorithmically. Follow these convenient steps:

1. Make sure there is no resource ID conflict between the 'NFNT's and 'sfnt's belonging to both families.
2. Make sure the style bits for italic are set in the font association table of “myFont italic.”
3. From ResEdit's File menu, “Get Info...” on the “myFont” 'FOND' resource. Write down the resource ID of the “myFont” 'FOND'.
4. From ResEdit's File menu, “Get Info...” on the “myFont italic” 'FOND'. Change its resource ID to be identical to the one you wrote down in step 3. Change its resource name to “myFont.”
5. Use the Finder in System 7 to move the contents of the “myFont italic” suitcase into the original “myFont” suitcase. It will merge all constituents into one font association table, and thus enable transparent substitution of the right font for QuickDraw's italic style.

Version Numbers

The 'FOND' structure (see *Inside Macintosh* Volume IV, page 45, “FamRec”) contains a field `ffVersion`, and inquiring minds naturally want to know more about it. Before anything else, however, please read the following disclaimer:

Disclaimer: The Font Manager does not check version numbers in a 'FOND', and we recommend that you not rely on the (intentionally vague) statements below, but rather analyze the data in the 'FOND' independently.

Currently, values 0...3 may appear in the `ffVersion` field, with the following intended interpretations:

- Version 0: Usually indicates that the 'FOND' has been created on the fly by the Font/DA Mover (or the System 7 Finder). But the 'FOND' for Palatino on the distribution disks of System 7 is a counterexample.
- Version 1: Obviously indicates the first version when 'FOND's came out (*Inside Macintosh* Volume IV, page 36).
- Version 2: Corresponds to the extension of the 'FOND' format documented in *Inside Macintosh* Volume V, page 185 (which does not mean that the 'FOND' actually contains a bounding box table).
- Version 3: The 'FOND' is supposed to contain a bounding box table.

This brings up an annoying fact. All measurement values (referring to a hypothetical 1-point font) in the 'FOND' are in a 16-bit fixed-point format, with an integer part in the high-order 4 bits and a fractional part in the low-order 12 bits. You would expect that negative values (like for `ffDescent`, or in the kerning tables) are represented in the usual two's-complement format, such that standard binary arithmetic applies. This is mostly true, but not always. Again, Palatino is a counterexample (and probably not the only one). To our knowledge, version 0 and version 1 'FOND's have negative values represented in a format where the most significant bit is the sign bit, and the rest represents the absolute value. However, there is nothing in the system software that enforces this, so counterexamples may exist.

Warning: Don't rely on the version number, but include sanity checks for the negative values in a 'FOND' instead! The following Pascal function shows how this can be done:

```
FUNCTION Check4p12Value(n: Integer): Integer;
{ n is a 4.12 fixed-point value; i.e., its "real" value is n/4096. }
{ If n is "unreasonably negative," interpret the most significant bit }
{ as sign bit, and convert to the usual two's complement format. }

BEGIN
  IF n < $8FFF THEN { means: (4.12-interpretation of n) is below - 7 }
    Check4p12Value := - BitAnd(n,$7FFF)
  { i.e., mask sign bit, and take negative of absolute value }
  ELSE
    Check4p12Value := n;
END;
```

In the Heart of the Font Manager

Swapping Fonts

As stated in *Inside Macintosh*, there is only one contact between QuickDraw and the Font Manager: the `FMSwapFont` function. Each of the three QuickDraw text *measuring* functions (`CharWidth`, `StringWidth` and `TextWidth`) always ends up in the

QuickDraw bottleneck procedure `QDProcs.txMeasProc`. Each of the three QuickDraw text *drawing* procedures (`DrawChar`, `DrawString` and `DrawText`) always ends up in the `QDProcs.textProc` bottleneck procedure. Any reasonable `textProc` (like `StdText`) needs to call the currently-installed text measuring bottleneck procedure before actually rendering the text. And what does any reasonable text measuring bottleneck procedure (like `StdTxMeas`) do first, before anything else? It calls `FMSwapFont`, to make

sure we are talking about the right font and its properties! (To be precise, `GetFontInfo` and `FontMetrics` are the other calls that make sure the right font is swapped in and set up, without requiring you to call `FMSwapFont` explicitly.)

Responding to a font request is a lot of work, and `FMSwapFont` has been optimized to return as quickly as possible if the request is the same as the previous one. Building the **global width table** (see *Inside Macintosh* Volume IV, page 41) is among the more time-consuming tasks related to `FMSwapFont`; this is why the Font Manager maintains a cache of up to 12 width tables.

Inside Macintosh Volume I, page 220 documents the Font Manager's choice when a font of the requested size is not available. However, some consequences or additional features have occasionally been a surprise to developers (and users as well).

Scaling Factors in `FMOutput` and `StdTxMeas`

Let's suppose you have only a 12-point bitmap version of Palatino, and don't have any Palatino outline fonts. When you request Palatino 18, QuickDraw sets up the `FMInput` record with `size = 18` and `numer = denom = Point($00010001)`. On return, the `FMOutput` record contains the handle to the font record to use (the 'NFNT' with the Palatino 12 bitmap font strike), and indicates the scaling factors QuickDraw will have to use to produce the desired text point size in `FMOutput.numer` and `FMOutput.denom`. In this example, that ratio is 3/2.

Note that these are also the values returned in `StdTxMeas` (*Inside Macintosh* Volume I, page 199) if you call the procedure with `numer = denom = Point($00010001)`. Why? Because `StdTxMeas` calls `FMSwapFont`, as explained under "Swapping Fonts." `StdTxMeas` does **not** apply these scaling factors to the text it measures. In our example, it would measure Palatino 12 and return `numer` and `denom` in the ratio 3/2 to tell you that your application must multiply the results by these values to get the correct measurements for Palatino 18. This has surprised more than one programmer who didn't expect `numer` and `denom` to change!

By the way, the Font Manager always normalizes the scaling factors as fractions `numer/denom` such that the denominator is equal to 256. In our example, the real numbers returned by `FMSwapFont` or `StdTxMeas` are `numer = 384` and `denom = 256`.

Warning: If the scaling factors `numer` and `denom` passed to `StdTxMeas`, `StdText` (see *Inside Macintosh* Volume I, pages 198 and 199), or in the `FMInput` record to `FMSwapFont` are such that `txSize*numer.v/denom.v` is less than 0.5 and rounds to 0, and if there is more than one 'sfnt' resource referenced in the font association table, then the current Font

Manager may get confused and return results for the wrong font strike.

TrueType Always Has the Right Size

The default value of `outlinePreferred` is `FALSE`. If you have bitmap fonts for Palatino 12 and Palatino 14 in your system as well as a Palatino TrueType font, then requests for Palatino 12 or Palatino 14 are fulfilled with the bitmap fonts, but requests for any other size are

fulfilled with the TrueType font. In particular, if you (or, for example, a printer driver) need Palatino 12 scaled by 2, the Font Manager will actually look for Palatino 24 and return the outline font, regardless of the setting of `outlinePreferred`. Even if you wanted the bitmap font doubled for exact “what-you-see-is-what-you-get” text placement, you’re out of luck—you get the TrueType font, which may have very different font metrics or character shapes.

If the Font Manager uses an outline font to fulfill a given font request, the `IsOutline` function returns `TRUE`. Interestingly, this does not imply that `RealFont` returns `TRUE` as well. If the text size is smaller than the value `lowestRecPPEM` (“smallest readable size in pixels”) in the 'head' font header in the TrueType font (see *The TrueType Font Format Specification*, version 1.0, page 227), then `RealFont` returns `FALSE`!

First Size, Then Style—or: To Be or Not to Be Outline

When the Font Manager walks the font association table of a 'FOND' to look for a font strike of a specified size and style, it stops at the first font of the right size. Only if you requested a stylistic variant (like bold or italic) does it take a closer look at the fonts of the same size. It does this by putting weights on the various style bits (for example, 8 for italic, 4 for bold, 3 for outline) and choosing the font strike whose style weight most closely matches the weight of the requested style. All this is fine when only bitmap fonts are available. With the presence of TrueType outlines, however, the results are not always as expected, depending on the font configuration installed.

Let’s look at a few examples:

Example 1: Let’s suppose you have the bitmap font Times 12 (Normal) and the TrueType fonts Times (Normal), Times Italic and Times Bold in your system. If you request Times 14 Italic or Times 14 Bold, it’s rendered from the Times Italic or Times Bold TrueType fonts. However, if you ask for Times 12 Italic or Times 12 Bold, and your system has the default setting of `outlinePreferred = FALSE`, the Font Manager decides to take the Times 12 bitmap and let QuickDraw algorithmically slant it (for italics) or smear it (for bold).

Example 2: Let’s suppose you want to draw big, bold Helvetica characters and there are no existing bitmaps for the size you want. If the Helvetica Bold TrueType outlines are available, the Font Manager chooses them and the only surprise in text rendering will be a pleasant one. If there is no Helvetica Bold TrueType font, however (like in the machine of your customer, who kept only the normal Helvetica TrueType font in his system), then the characters are rendered using the normal Helvetica outlines and, in a second step, QuickDraw applies its horizontal 1-pixel “smearing” to simulate the bold stylistic variant. The result is very different (and rather an unpleasant surprise).

Example 3: Admittedly, this is less likely (but it has happened). Let's suppose somebody decides to rip the Times TrueType outline out of the System file (don't ask me why—I don't know). He forgets to take the Times Italic TrueType outline away as well. The next time he draws text in Times (Normal), in a size for which there is no bitmap font (or if `outlinePreferred = TRUE`), the Font Manager goes for an 'sfnt', and the text shows up in *italic* (what a surprise!).

Unfortunately, given the current implementation of the Font Manager, there are no solutions to the problems illustrated above—other than asking users of your application to install the fonts you recommend. The only way to anticipate these potential surprises from within your application is to look into the 'FOND's font association table. You can't depend on the `IsOutline` function because it returns `TRUE` as soon as the Font Manager stops at an 'sfnt', in its first pass through the font association table—regardless of subsequent stylistic variations. This means, for example, if you ask for Helvetica Bold and `IsOutline` returns `TRUE`, you don't know if you got the Helvetica Bold TrueType font or if QuickDraw “smeared” the Helvetica (Plain) TrueType font.

Where Do the Widths Come From?

Text measuring (for example, for precise text placement in forms with bounding boxes) and most line layout algorithms for justified text rely heavily on the character widths contained in the global width table. Given that under the current font architecture, we may easily have three or more different width tables for the same font specification (the non-proportional integer widths attached to the 'NFNT', the fractional widths contained in the 'FOND', and the fractional widths provided by the 'sfnt'), it is important to understand where the widths come from in any case.

Since `SetFractEnable` was introduced (*Inside Macintosh* Volume IV, page 32 and Volume V, page 180), its setting `TRUE` or `FALSE` was supposed to give predictable effects. If it's `FALSE`, the Font Manager takes the integer widths from the 'NFNT'; if it's `TRUE`, it takes the fractional widths from the 'FOND'. Unfortunately, there are some additional details and side effects that are not well known.

- The Font Manager looks at bit 14 of the `ffFlags` field in the 'FOND' (see *Inside Macintosh* Volume IV pages 36 and 37). If it is set (like it is for Courier), the fractional widths from the 'FOND' are *never* used.
- If `SetFractEnable` is `TRUE` and you request a stylistic variation like bold or italic, the Font Manager looks at bits 12 and 13 of the `ffFlags` field to decide how different widths or extra widths for the stylistic variants have to be used. What it decides is documented in the “Font Manager” chapter of *Inside Macintosh Preview*, located on the *Developer CD Series* discs.
- Given that it is not possible to set the pen to a fractional position, precise text positioning with fractional widths enabled is always compromised because of (accumulated) rounding errors.
- QuickDraw distributes the accumulated rounding errors across characters within a string (instead of adding it at the end of the drawn text). This results in poor text quality on the screen, and in problems when calculating the position of the insertion point between characters.
- The LaserWriter driver watches what you pass to `SetFractEnable`. Passing `TRUE` to `SetFractEnable` disables some of the LaserWriter driver's line layout features, assuming

that the programmer intends to control text placement manually. Explicitly passing FALSE to SetFractEnable achieves different results than using the default value of FALSE—Font Substitution behaves differently, for example. These effects are sometimes Not What You Wanted.

- On non-32-Bit-QuickDraw systems, SetFractEnable is not recorded in pictures. This affects the line layout of text reproduced through DrawPicture if the picture was created with fractional widths enabled.

In systems with TrueType, quite naturally the widths *always* come from the 'sfnt' when the Font Manager uses a TrueType font. If `fractEnable` is `FALSE`, hand-tuned integer character widths for specific point sizes come from the 'hdmx' table in the 'sfnt'. If `fractEnable` is `FALSE` and no 'hdmx' table is present or it contains no entries for the desired point size, the fractional character widths from the 'sfnt' are rounded to integral values.

More Line Layout Problems

The routines `SpaceExtra` (*Inside Macintosh* Volume I, page 172) and `CharExtra` (*Inside Macintosh* Volume V, page 77; available only in color GrafPorts) are intended to help you draw fully justified text. This works fine on the screen, but not all printer drivers are smart enough to use these settings appropriately under all circumstances. In particular, if you pass `TRUE` to `SetFractEnable`, or if you turn the LaserWriter driver's line layout algorithm off (by means of the picture comment `LineLayoutOff`; see Macintosh Technical Note #91), or if font substitution is enabled and actually occurs, it is better not to rely on `SpaceExtra` and `CharExtra` when printing fully justified text. Instead, keep the LaserWriter driver's line layout adjustments off, and calculate the placement of your text (word by word, or even character by character) yourself.

Putting Text Into Boxes

TrueType fonts came to the Macintosh together with seven new Font Manager routines (as documented in *Inside Macintosh* Volume VI, Chapter 12). The `OutlineMetrics` function is certainly the most sophisticated of these, and sample code illustrating its usage may be helpful. The following procedure `DrawBoxedString` assumes that the new outline calls (*Inside Macintosh* Volume VI, Chapter 12) are available, and that `IsOutline` returns `TRUE` for the current port setting.

```
PROCEDURE DrawBoxedString(pt: Point; s: Str255);
{ Draw string s at pen position (pt.h, pt.v), and show each character's bounding box. }

CONST
    kOneOne = $00010001;

VAR
    advA: FixedPtr;
    lsbA: FixedPtr;
    bdsA: RectPtr;
    err,i,yMin,yMax,leftEdge,temp: Integer;
    numer,denom: Point;
    advance,lsb: Fixed;
    r: Rect;

BEGIN
    numer := Point(kOneOne);
    denom := Point(kOneOne); { unless you want to draw with scaling factors
        .... }

    MoveTo(pt.h,pt.v);
    DrawString(s);
{ This is for the pleasure of your eyes only - in practice, you would probably }
{ first look at the metrics, and then decide where and how to draw the string! }
    advA := FixedPtr(NewPtr(Length(s) * SizeOf(Fixed)));
    lsbA := FixedPtr(NewPtr(Length(s) * SizeOf(Fixed)));
    bdsA := RectPtr(NewPtr(Length(s) * SizeOf(Rect)));
    { Please, check for NIL pointers here! }
    err := OutlineMetrics(Length(s),@s[1],numer,denom,yMax,yMin,advA,lsbA,
        bdsA);

    advance := 0;
    FOR i := 1 TO Length(s) DO { for each character }
        BEGIN
            { Add accumulated advanceWidth and leftSideBearing of current glyph }
            { horizontally to starting point. }
            leftEdge := pt.h + Fix2Long(advance + lsbA^);
```

```

    r := bdsA^; { The bounding box rectangle is in TrueType coordinates. }
    temp := r.bottom; { need to flip it "upside down" }
    r.bottom := - r.top;
    r.top := - temp;
    OffsetRect(r, leftEdge, pt.v);
    FrameRect(r); { This is the glyph's bounding box. }
    advance := advance + advA^;
    { "Advance" is Fixed, to avoid accumulation of rounding errors. }
    { Now, bump pointers for next glyph. }
    bdsA := RectPtr(ord4(bdsA) + SizeOf(Rect));
    advA := FixedPtr(ord4(advA) + SizeOf(Fixed));
    lsbA := FixedPtr(ord4(lsbA) + SizeOf(Fixed));
END;
DisposPtr(Ptr(advA));
DisposPtr(Ptr(lsbA));
DisposPtr(Ptr(bdsA));
END; { DrawBoxedString }

```

OutlineMetrics exists because many developers need pixel-precise information on placement and bounding boxes, often on a character-by-character basis. Unfortunately, there is no similar facility for text drawing with bitmap fonts. Worse, under certain circumstances, italicized or shadowed (or both) bitmap fonts are sometimes poorly clipped, particularly for scaled sizes. Cosmetic workarounds include adding a space character to strings drawn in italic. You might also draw the text off-screen first (in order to determine the bounding box of the black pixels) and use CopyBits to copy the text onto the screen—but using CopyBits for text is usually bad for printing.

The existing documentation on the FMOutput and global width table structures (*Inside Macintosh* Volume I, page 227 and Volume IV, page 41) suggests it's possible to devise a routine for determining a fairly precise text bounding box for bitmap fonts. The procedure below, BitmapTextBoundingBox, is a first attempt. It assumes that TrueType is unavailable, or that the IsOutline call returned FALSE for the current port settings. While the returned bounding box is not always “tight,” be careful before modifying the algorithm and shrinking the resulting bounding box—bitmap fonts just don't contain enough precise information for an exact bounding box, and different bitmap fonts and different sizes may require different adjustments.

```

PROCEDURE TextBoundingBox(s: Str255; numer,denom: Point; VAR box: Rect);

CONST
    FMGrOutRec = $998; { FMOutRec starts here in low memory }
    tabFont = 1024;
    { global width table offset for font record handle, see IM IV-41 }

TYPE
    FontRecPtr = ^FontRec;

VAR
    hScale,vScale: Fixed;
    err,intWidth,kernAdjust: Integer;
    xy: Point;
    info: FontInfo; { only for StdTxMeas; we'll use FontMetrics }
    fm: FMetricRec; { see Inside Macintosh, IV-32 }
    fmOut: FMOutput;
    h: Handle;

BEGIN
    intWidth := StdTxMeas(ord(s[0]),@s[1],numer,denom,info);
    { calls FMswapFont and everything - }
    { StdTxMeas returns possibly modified scaling factors numer, denom }
    hScale := FixRatio(numer.h,denom.h);

```

```
vScale := FixRatio( numer.v, denom.v );
{ These are the scaling factors QuickDraw uses      }
{ in "stretching" the available character bitmaps }
fmOut := FMOutPtr( FMgrOutRec^ );
{ has been filled by the most recent FMswapFont,   }
{ implicitly called by StdTxMeas                   }
SetRect( box, 0, - info.ascent, intWidth, info.descent );
{ bounding box for unscaled plain text              }
IF ( italic IN thePort^.txFace ) AND ( fmOut.italic <> 0 ) THEN BEGIN
{ the following is heuristics ... }
    box.right := box.right + ( info.ascent + info.descent - 1 ) *
                    fmOut.italic DIV 16;
    FontMetrics( fm );
    HLock( fm.WTabHandle ); { We'll point to global WidthTable. }
    h := Handle( LongPtr( ord4( fm.WTabHandle^ ) + tabFont^ ) );
    { Be sure it's a handle to a 'NFNT' or 'FONT' ! }
    kernAdjust := FontRecPtr( h^ )^.kernMax;
    OffsetRect( box, - kernAdjust, 0 );
    HUnlock( fm.WTabHandle );
END;
IF ( bold IN thePort^.txFace ) AND ( fmOut.bold <> 0 ) THEN
    box.right := box.right + fmOut.bold - fmOut.extra;
IF ( outline IN thePort^.txFace ) THEN InsetRect( box, - 1, - 1 );
IF ( shadow IN thePort^.txFace ) AND ( fmOut.shadow <> 0 ) THEN BEGIN
    IF fmOut.shadow > 3 THEN fmOut.shadow := 3;
    box.right := box.right + fmOut.shadow;
    box.bottom := box.bottom + fmOut.shadow;
    InsetRect( box, - 1, - 1 );
END;
{ Now scale the box (more or less) as QuickDraw would do. }
{ Note that some of the adjustments are based on trial and error... }
box.top := FixRound( FixMul( Long2Fix( box.top ), vScale ) );
box.left := FixRound( FixMul( Long2Fix( box.left ), hScale ) ) - 1;
box.bottom := FixRound( FixMul( Long2Fix( box.bottom ), vScale ) ) + 1;
box.right := FixRound( FixMul( Long2Fix( box.right ), hScale ) ) + 1;
GetPen( xy );
OffsetRect( box, xy.h, xy.v );
END;
```

Conclusion

At the time when the original Font Manager architecture was designed, based on QuickDraw's hard-coded 72 dpi resolution, nobody could anticipate that some years later, the Macintosh would be used to tackle professional typesetting projects. Several advanced page layout applications managed to work around the "built-in" limitations, at high development costs, and some compatibility and performance problems. In many other cases, however, those limitations caused questions to DTS and unsatisfying compromises. This Note can't do much more than explain the state of affairs; the real solution to the problems must come from a redesigned foundation. TrueType leads the way and already fulfills many of the requirements; everything else is getting closer and closer.

Further Reference:

- *Inside Macintosh*, Volume I, Chapter 7, The Font Manager
- *Inside Macintosh*, Volume IV, Chapter 5, The Font Manager
- *Inside Macintosh*, Volume V, Chapter 9, The Font Manager
- *Inside Macintosh*, Volume VI, Chapter 12, The Font Manager
- *New & Improved Inside Macintosh*, Imaging: The Font Manager. *Developer CD Series* disc, path Developer Essentials: Technical Docs: Inside Macintosh Previewf
- Macintosh Technical Note #91, Picture Comments—The Real Deal
- M.IM.FontNames
- M.TE.FontsAndScripts
- M.IM.FontFamilies
- *Apple LaserWriter Reference*, Chapter 2, Working With Fonts (Addison-Wesley, 1988)
- Adobe Technical Note #0091 (PostScript Developer Support Group), Macintosh FOND Resources

PostScript and Adobe are registered trademarks of Adobe Systems Incorporated.
Helvetica and Palatino are registered trademarks of Linotype AG and/or its subsidiaries.

Velocio is **not** a trademark of the author.