

New Technical Notes

Macintosh



®

Developer Support

Font Manager Q&As

Text

Revised by: Developer Support Center

June 1993

Written by: Developer Support Center

October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you’ve sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don’t have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As in this Technical Note:

Macintosh fonts that are likely to be available

Macintosh fonts that are likely to be available

Date Written: 2/3/93

Last reviewed: 4/1/93

Starting with System 6.0.5, are there certain guaranteed fonts that are available regardless of the language system running? I’m hoping that Geneva 9, Geneva 12, and Chicago 12 will always be available for my cdev.

There’s no such thing as a guaranteed font. Since fonts are simply resources in the System file or in the Fonts folder, users *can* remove or replace them with different ones if they’re sufficiently determined. Apple makes no guarantees the system will work if they do, but it’s possible.

There are resources in the System file that tell both Font/DA Mover and the System 7 Finder not to show specific fonts so the users can’t easily remove them without a resource editor. Font/DA Mover’s resource is of type FRSV and contains font IDs that correspond to

Chicago 12, Geneva 9, Geneva 12 and Monaco 9. The System 7 resource type is 'resf' and it contains font names and point sizes (such as not to rely upon ID numbers) and it also protects Chicago 12, Geneva 9, Geneva 12 and Monaco 9. Chicago 12 is usually in ROM as well, so it's a lot harder to remove.

So you have a pretty decent chance that these four fonts are available. Even if they're not, though, you'll get *some* font so text will draw even if it doesn't fit in the space you've provided. Be careful about extremely tight text spaces—it makes it a lot harder to internationalize your products since non-English text can expand by an average of 50 percent.

Chicago Control-Q prints propeller or clover symbol

Date Written: 7/4/90

Last reviewed: 8/1/92

How do I get the character that represents the clover used for command-key equivalents in documents and in menus?

—

This key is documented in the Apple Style Guide, which is available on the latest *Developer CD Series* disc as well as from APDA. One little feature of Key Caps which is not widely known is the Control key (not the Command or Option keys). Pressing the Control key in Chicago shows that Control-Q in Chicago maps to the propeller symbol for which you search. Control-Q generates the character code 17; the standard Macintosh character set (see *Inside Macintosh* Volume VI, page 12-5) specifies this symbol for it.

Determining PostScript font intrinsic styles

Date Written: 1/16/91

Last reviewed: 8/1/92

How can I determine the intrinsic styles supported by a particular PostScript font?

—

To find the styles that are supported by a particular font, you need to access one of two tables. To find the screen fonts that are available, you can use the font association table, shown on page 38 of *Inside Macintosh* Volume IV. Each entry in the table contains the size and style of the font, as well as the resource ID. Each font in this table should be available for the screen.

When you print a font to a PostScript printer, the Font Manager uses the style-naming table in the family record to create a PostScript name for the font. The table has a list of strings, followed by an entry for each style supported by that particular font. A table for Times might look like this:

```
Value  Description  Hex Dump
9  Number of strings  <INTEGER>  09
"Times"  Basename of font  05, 54, 69, 6D, 65, 73
6, 7  Suffix index for style 1  02, 06, 07
Pascal string that looks like:
String[0] := CHR(2);
String[1] := CHR(6);
```

```
String[2] := CHR(7);  
6, 8 Suffix index for style 2 02, 06, 08  
6, 9 Suffix index for style 3 02, 06, 09  
6, 8, 9 Suffix index for style 4 03, 06, 08, 09  
"-" Suffix 1 01, 2D  
"Roman" Suffix 2 05, 52, 6F, 6D, 61, 6E  
"Bold" Suffix 3 04, 42, 6F, 6C, 64  
"Italic" Suffix 4 06, 49, 74, 61, 6C, 69, 63
```

Except for the first entry, all entries in the table are stored as Pascal strings with a length byte followed by one byte for each character in the string.

The table is used to build the PostScript font name for a particular style. This is a little complicated, so hang on. Let's look at the table again, with each line numbered, and without all the hex:

```
0:  9
1:  "Times"
2:  6, 7
3:  6, 8
4:  6, 9
5:  6, 8, 9
6:  "-"
7:  "Roman"
8:  "Bold"
9:  "Italic"
```

Entry #0 is the number of strings in this table. Entry #1 is the basename of the font. Entries #6-9 are the suffixes that will be appended to the basename for particular styles. Entries #2-5 are the indexes of the suffixes required to create the PostScript names for the different styles (entry 2 = plain, 3 = bold, 4 = italic, and 5 = bold+italic).

Let's use entry #2 to build a name:

1. Always start with the basename: "Times"
2. Append the first suffix, #6 in this case: "Times-"
3. Append the second and last suffix, #7: "Times-Roman"

The style-naming table is not documented in the Font Manager chapter of *Inside Macintosh* because it's too specific to PostScript. Instead, it's documented in the LaserWriter Reference (Addison-Wesley) on pages 28-35. The manual's description doesn't include examples, so the above example makes things easier to understand.

Determining Macintosh system font size

Date Written: 2/11/91

Last reviewed: 8/1/92

How does a program determine the default system font size?

—

If you want to know the default system font size, use the `GetDefFontSize` call (*Inside Macintosh* Volume V, page 314). This call will return the true default system font size. On all Roman systems the `sysFontSize` low-memory global is always zero by default; this means that it's actually 12. Don't ask me why they did this, but it's true, so you should use the call if you want a painless method of obtaining this information. Try to avoid reading the global directly.

FScaleDisable and Macintosh screen rendering

Date Written: 6/4/91

Last reviewed: 8/1/92

What is the current interface guideline on font scaling? Should we set FScaleDisable to true or false for screen rendering? I believe the old guideline was to set it to true (thus disabling font scaling) so that if you have a 23 pt screen font you will see 12 pt glyphs with 23 pt widths. We want performance on Apple's low-end machines, and we don't want to make this a user preference.

Most applications now set FScaleDisable to false all the time. If you are worried about the speed degradation in your application when running on slower Macintosh systems, you could have your software identify which Mac it is running on using Gestalt, SysEnvirons, etc., and then set FScaleDisable to true only on the slow machines. From the purely interface point of view, "what-you-see-is-what-you-get" is best, if processor speed allows it.

With TrueType there is no issue, since FScaleDisable doesn't have any effect on an outline font.

Macintosh double-byte character encoding

Date Written: 8/5/91

Last reviewed: 8/1/92

When will System 7 and TrueType be able to support a larger character encoding vector than the current 256 characters? Is this something I could do now?

System 7 does not have double-byte character encoding support now. In the future, Apple will use the two-byte UNICODE standard for its operating systems. However, such a change will be huge and therefore will not be available in the near future.

Ever since 6.0.2, KanjiTalk has had double-byte encoding, so you can use the Japanese system at this time. The system which allows you to do this is called shift-JIS (Japanese Input System). The mappings are on the current Developer CD in the Kanji folder.

'FOND' resource features subject to change

Date Written: 6/17/91

Last reviewed: 8/1/92

Which 'FOND' features are subject to change, and what parts can I rely on?

'FOND' features not documented in *Inside Macintosh*, Macintosh Technical Notes, or *develop* are subject to change. That's the official word.

Five font style-mapping table styles

Date Written: 6/17/91

Last reviewed: 8/1/92

Why is “47” the bounds of the indexes’ array in the StyleTable?

—

The style-mapping tables, which are used for mapping of a font style to a particular font for printing, have only five possible styles, unlike the six screen styles (underline is omitted, since a laser printer just explicitly draws a line under the string—it doesn't need a special font for that). You can have any combination of these five styles, EXCEPT that you can't have "Condense" and "Extend" at the same time (Condensed Extended wouldn't make a whole lot of sense). Thus you get (combinatorics come back to haunt us) 48 possible unique styles to map to.

X-Ref:

Snippet "StyleMap" on the Developer CD.

LaserWriter Reference, Chapter 2, "Working with Fonts," (Addison-Wesley; APDA #M7073, \$19.95).

System 7 and modified fonts

Date Written: 6/21/91

Last reviewed: 8/1/92

We ship modified versions of the Chicago and Geneva fonts and their FONDS in our application's resource fork. With System 6, when we ask for font 0 or font 3, we get our modified fonts in windows, buttons, menus and dialogs, but not with System 7. Is there a new way to tell System 7 to use my version of Chicago 12 to display all system related stuff?

What's happened is that the system software is being much more strict about whose "Chicago" it uses for menus and dialogs. The Menu and Control Managers now only look at the system file for the Chicago they use.

There are a couple of ways to get around this: First, you can try patching DrawString right before calling MenuSelect in your program. The patch would select your Chicago and then jump to the standard DrawString. After MenuSelect, remove the patch. The disadvantage of this method is that, if a future system software release doesn't use DrawString for drawing menus, the patch would cease to have any effect.

A better solution is to write your own menu and control definition code—in other words, your own custom MDEF and CDEFs. The way you typically do this is to get a copy of the standard Macintosh system's MDEF or CDEF, and alter it to your specifications. In your case, this would be merely selecting your font instead of the system font. Name your font something other than Chicago and just select that font by name in the menu or control's draw routine. The Control and Menu Manager chapters in *Inside Macintosh* Volume I have more information on writing custom definitions. There is only one problem with this right now: While the System 6 MDEF and CDEF are available on AppleLink, the System 7 versions are not available yet, although they will be soon.

A note on the use of Chicago in your application: As the June 1991 edition of the Macintosh

Technical Note “Font Family Numbers” mentions, fonts are copyrighted material. Apple owns the Chicago font and typeface, so be sure you check into licensing issues before releasing any version, altered or unaltered, with your application.

Spanish typographic measurements

Date Written: 12/10/91

Last reviewed: 8/1/92

What typographical measurement issues must be considered for Spanish systems? Do the Spanish specify type in ciceros and didots instead of points?

—

If typesetting is done in Spain with computers, U.S. standards generally are used. It is only when typesetting is done the old-fashioned way that you'll see different measurements.

Paper sizes are different. In Spain DIN-44 (210 x 297), DIN-A3 (420 x 297) and "folio" (215 x 315) are used. As always, you'll be working with 72 dpi for the screen (and any time you use QuickDraw) but something different on paper, so you'll need to use PrGeneral and image the stuff yourself to a resolution that allows you control over your imaging for printing. This is detailed in the article, "Meet PrGeneral, the Trap That Makes the Most of the Printing Manager," in issue #3 of *develop*.

Here are the measurements used in Spain:

- Decimal point (Didot) (0.3759 mm)
- Millimeter
- Cicero (12 decimal points)
- Centimeter
- Inches

SetOutlinePreferred = TRUE or not?

Date Written: 12/2/91

Last reviewed: 8/1/92

My application calls SetOutlinePreferred so that TrueType fonts are used if both bitmapped and TrueType fonts are in the system. It was reported to me, however, that some international TrueType fonts look really bad at small point sizes on the screen. Should I avoid calling this function?

—

SetOutlinePreferred is best used as a user-selectable option. Along the same lines, you might want to include the SetPreserveGlyph call (*Inside Macintosh* Volume VI, page 12-21)—again, as a user-selectable option.

Currently, the default for outlinePreferred is FALSE for compatibility reasons (existing documents don't get reflowed if the bitmapped fonts are still around) and for aesthetic and performance reasons (users are free to maintain bitmapped fonts in the smaller point sizes if the TrueType version isn't satisfying for small sizes or is too slow). On the other hand, as soon as a bitmapped font is unavailable for a requested point size, and a TrueType font is present, the TrueType font is used even with outlinePreferred = FALSE. Setting

`outlinePreferred = TRUE` makes a difference only for point sizes where a bitmapped font strike is present along with an 'sfnt' in the same family.

TrueType fonts might be preferable even for small point sizes if linearly scaled character widths are more important than screen rendering: if the main purpose of a program is preprint processing for a high-resolution output device, `outlinePreferred = TRUE` may give better line layout results on the printer, at the price of “not so great” type rendering on a 72 dpi screen.

(An example of the conflict between linearly scaled TrueType and nonlinearly scaled bitmapped fonts is Helvetica: `StringWidth('Lilli')` returns 19 for the 12-point bitmapped font, and 15 for the 13-point size from TrueType!)

All this boils down to the recommendation stated initially: the user should be given the flexibility to decide whether to use the existing bitmaps (using TrueType only for bigger point sizes and high-resolution printers), or to go with TrueType even if the result on the screen is not optimal. (By the way, it's likely that TrueType development will substantially reduce this conflict in the future.)

FontRec fontType field and determining monospaced fonts

Date Written: 1/13/92

Last reviewed: 8/1/92

How can I tell whether a font is monospaced or proportional? The FontRec record's fontType field doesn't correctly tell me whether the font is fixed width as *Inside Macintosh* Volume V says it should. All system fonts appear to have the same fontType regardless of whether they're fixed or proportional. Currently I test whether the width of the characters "m" and "i" are equal and if they are, I consider the font to be fixed width. Is there an easier (and faster!) way?

—

The Font Manager documentation is not explicit enough about the fact that bit 13 (0x2000) of the fontType field is useless. The Font Manager doesn't check the setting of this bit, nor does QuickDraw (or any printer driver). As you observed, monospaced fonts like Monaco or Courier don't have the bit set; the bit is meaningless. In addition, the fontType field is available only for 'FONT' and 'NFNT' resources; it does not exist in 'sfnt' resources, and you would have to check separately for the resource type of the font. Your idea of comparing the widths of "m" and "i" (or any other characters that are extremely unlikely to have the same widths in a proportionally spaced font) is indeed the only reasonable way of figuring out whether a font is monospaced.

Getting global width table for a font specification

Date Written: 4/22/92

Last reviewed: 8/1/92

What's the fastest way to get the width table for a given font? FontMetrics is too slow, especially in color. Is there any other call that will get the global width table set up correctly? That is all that we need from the call to FontMetrics.

—

FontMetrics does not have much overhead in setting up the width table. It does a dummy `DrawChar(' ');` this is very rapidly transformed into a call to `StdText`, and `StdText`

immediately calls StdTxMeas. The first thing StdTxMeas does is to set up the input parameters for a FMSwapFont call and call it. FMSwapFont is the heart of the Font Manager, and does all the work. It comes back with the FMOutput record, the font strike, and the width table. From there, FontMetrics derives the values it needs to bring back.

You probably wouldn't save much more than 1/1000th of a second if, instead of calling FontMetrics, you called FMSwapFont explicitly yourself—our only alternative suggestion.

The Font Manager does quite a lot of caching (up to 12 width tables), and managing the cache takes some cycles, too. If there is nothing in the cache corresponding to the font request, the cache makes the call even slower than it would be without cache.

The next source of overhead is the Resource Manager. Looking for a specific font involves going through the whole resource chain first for the FOND (if none is found, the search restarts for a FONT), and then, based on the FOND's font association table, for a NFNT or 'sfnt'. If no NFNT is found, the search restarts for a FONT, always through the whole resource chain. For a huge resource fork like in the System file (and, maybe, also in your application), the time spent in the Resource Manager is not negligible—in particular, if you have add-ons in your system (such as INITs or 'cdev's) that patch out Resource Manager calls, maybe several times, and usually slow it down considerably!

Even in case this hurdle is overcome swiftly (after all, the Resource Manager has its own caching scheme for optimization), the next step necessarily takes some time, and, as you have observed, especially on a color system: It consists of actually providing the bitmap for the font strike. If the screen depth is >1 , this involves creating “synthetic” fonts for the correct screen depth, to optimize text drawing. Also, if the font is an outline font, the first time a font strike has to be rasterized is quite costly in terms of machine cycles.

Finally, the width table can be created; and, because of the scaling factors involved, this requires 256 times some arithmetic which is known never to be fast enough.

All this certainly gives us an understanding for the time it takes FMSwapFont (FontMetrics) to get the job done, but it does not solve your problem.

Depending on how predictable the usage of fonts and width tables in your application is, you might consider building kind of a database of width tables beforehand, or along the way, and use this information directly from within your application. There is no shortcut at all through the Resource Manager to get at the font resources, and there is no shortcut within FMSwapFont, like not building the font bitmaps. (To the best of my knowledge, the needbits field in the FMInput record does *not* have this effect.) The only obvious way to get width tables faster is to keep them around, and to extend manually the capacity of the Font Manager's cache of width tables.

SetFractEnable and recalculating width tables

Date Written: 5/5/92

Last reviewed: 8/1/92

Calling SetFractEnable seems to force the width tables to be recalculated regardless of the setting of the low-memory global FractEnable. We're calling this routine at a central entry point for any document, as it's a document-by-document attribute. We then unconditionally call SetFractEnable(false) on exit back to the event loop, to be nice to other applications.

Calling SetFractEnable(false) seems to trigger the recalculation even though FractEnable is false. What's the best way to get around this?

—

Your observation is correct. The SetFractEnable call stuffs the Boolean parameter (as a single byte) into the low-memory global \$BF4 and indiscriminately invalidates the cached width table by setting the 4-byte value at \$B4C (LastSpExtra, a Fixed value) to -1. Obviously, it wasn't

anticipated that SetFractEnable could be called regularly with a parameter that often doesn't change the previous setting. (By the way, the same observation applies to SetFScaleDisable.)

In your case, you may want to keep track of the fractEnable setting in your application and avoid redundant SetFractEnable calls. (Note that it's not a good idea to use the above insider information and poke at \$BF4 and \$B4C on your own!)

You don't need to think of other applications when resetting fractEnable; it belongs to those low-memory globals that are swapped in and out during context switches to other applications.

Corrupted Macintosh font or font suitcase criteria

Date Written: 6/19/92

Last reviewed: 11/30/92

How can I detect whether a font suitcase is corrupted when it's opened and whether any of the fonts in it are corrupted before any of the fonts are used? I know that the Finder is able to do this, and I was wondering whether Apple gives out this information. My program will run only under System 7, if that helps.

The Finder and font architecture on the Macintosh are living things; the definition of what is and isn't a damaged suitcase can change from release to release of system software. However, any of the following conditions makes System 7 report the suitcase as "damaged":

- More than eight FONDs reference the same font.
- The Finder can't create a new standalone icon object for each font in the suitcase. The usual cause of this is that two FONDs have the same name for the first 31 characters, and the Finder thinks there's already an icon in that window with the same name. (Two icons in the same directory with the same name is a sign of damage.)
- There isn't at least one font association table entry, or the table goes past the logical end of the resource.
- The first resource name in the map is zero-length. (This is a test for some older third-party corrupted suitcases.)
- The FOND has no name.
- The FOND doesn't have a valid character range; ffFirstChar has to be less than ffLastChar, unless it's a "dummy" FOND (created on the fly for old standalone FONTs, in which case ffLastChar is 0).
- All the font association table entries aren't in ascending point size order.
- Two font association table entries reference exactly the same point size and style.
- The offsets to the width table, kerning table, and style mapping table are invalid and nonzero.
- The font ID is 0 and it's not the system font.

We can't promise that this list is complete, but it does contain most conditions for which the Finder would report a suitcase as damaged.