New Technical Notes

Macintosh

Developer Support

Font Names

Imaging

Revised by: Bryan Stearns August 1988
Written by: Darin Adler April 1988

This note recommends the use of font names rather than font numbers.

The Font Manager chapter of *Inside Macintosh Volume IV* claims that font family numbers 0 through 127 are reserved for use by Apple, and numbers 128 through 255 are assigned by Apple for fonts created by software developers. This is no longer true. Developer Technical Support does **not** assign font family numbers. You should only use font numbers to reference the system font (font 0) and application default font (font 1). All other fonts should be identified by name. The Font/DA Mover will renumber a font when moving it into a file containing a conflicting font family.

The Font Manager routines `GetFontName` and `GetFNum` map font names to numbers and vice versa. This makes it simple to store a font's name in a document and turn it back into a number when reading the document. Unfortunately, `GetFNum` returns a 0 when a font by that name doesn't exist; this is the same as the font ID for the system font. The following routine in MPW Pascal alleviates this problem:

```
FUNCTION GetFontNumber(fontName: Str255; VAR fontNum: INTEGER) : BOOLEAN;
{GetFontNumber returns in fontNum the number for the font having
 the given fontName. If there's no such font, it returns FALSE.}

VAR
    systemFontName: Str255;

BEGIN
GetFNum(fontName, theNum);
IF fontNum = 0 THEN BEGIN
    {either the font was not found, or it is the system font}
    {if it was the system font, we got it, otherwise we didn't}
    GetFontName(0, systemFontName);
    GetFontNumber := EqualString(fontName, systemFontName, FALSE, FALSE);
END ELSE
    {if theNum was not 0, we found the font}
    GetFontNumber := TRUE;
END;
```

In MPW C:

```
Boolean GETFONTNUMBER(fontName, fontNum)
Str255* fontName;
short* fontNum;
/* GetFontNumber returns in fontNum the number for the font having
   the given fontName. If there's no such font, it returns false. */
{
    Str255* systemFontName;

    GETFNUM(fontName, fontNum);
    if (*fontNum == 0) {
        /* either the font was not found, or it is the system font */
        /* if it was the system font, we got it, otherwise we didn't */
        GETFONTNAME(0, systemFontName);
        return (EQUALSTRING(fontName, systemFontName, false, false));
    } else
        return true;
}
```

This routine makes it easy to find out if a given font exists; if the font isn't available, you can present a dialog to the user, allowing an appropriate substitute font to be chosen.

Handy Hint for Lists of Font Names

Most applications that offer the user a choice of fonts do so by creating a Fonts menu. Some applications, however, present a list of fonts in some other way: for example, word processors that use a dialog box to let the user pick a font family, point size, and style often display the font family choices in a List Manager list. You can get the Menu Manager to do most of the work by using `AddResMenu` to enumerate and alphabetize the names, as follows:

```
PROCEDURE BuildMyFontList;
CONST
       AnUnusedMenuID = 150; {a menu ID not used by any of your menus}
VAR
       tempMenu: MenuHandle;
       thisItem: INTEGER;
       aFontName: Str255;
BEGIN
       {Get a menu; use the Menu Manager to fill it with font names}
       tempMenu := NewMenu(AnUnusedMenuID,'x');
       AddResMenu(tempMenu,'FONT');

       {Extract the names we got, one at a time}
       FOR thisItem := 1 TO CountMItems(tempMenu) DO
              BEGIN
                     {Extract the next name from the menu}
                     GetItem(tempMenu,thisItem,aFontName);

                     {** Do something with this font name (add   **}
                     {** it to a List Manager list, or whatever) **}
              END;

       {We're done with the menu; dispose of it}
       DisposeMenu(tempMenu);
END; {BuildMyFontList}
```

This approach will help to insulate your application from changes to the Font Manager. Historically, `AddResMenu` was modified to notice `FOND` resources at the same time the Font Manager began to support them, so applications that used this technique to build their font lists didn't need to be modified to work with `FOND`s.

Suggested Font Strategy for Applications and Documents

If your application offers the user a choice of a single font for use in the entire document, it's a simple matter to store the name of that font somewhere in the document (perhaps in an 'STR ' resource). However, if your application lets the user use many fonts within each document (as is the case with most word processors) a more complex strategy is necessary: the Font Name Mapping Table. Each entry might look like this:

```
FontMapEntry : RECORD
       name: Str255;        {The name of the font}
       localID: INTEGER;    {a unique number for this entry}
       realID: INTEGER;     {last time we checked, this font's font number}
       useCount: INTEGER;   {How many times this font is used in this
                            document}
END;
```

In a new document, start out with no entries in the table. When the user changes a selection of text to a new font (that is, one whose name is not in the table), add an entry to the table. Set `useCount` to 1 (as this font is now referenced once within the document), and pick a `localID` that is unique within the table. In the text, or wherever you would normally keep the font number, store a copy of this `localID` instead of the font number. Use `GetFontNumber` (the example above) to get the "real" font number, and store it in the table as well, in `realID`.

Whenever you need to draw text, search through the table for the proper `localID`. When you find it, use the `realID` that is stored in that entry in a call to `TextFont`, then draw your text as usual.

Keep the `useCount` updated, so that you know when to get rid of a table entry. If the user deletes a range of text, or changes it to another font, examine the text to see if you should decrement any of the `useCount`s in your table. When a `useCount` for an entry becomes zero, you'll know that the font for that entry isn't used anywhere within the document, and you can remove the entry from the table.

When you save the document, save the table with it. When you open an existing document, load the table. For each entry, call `GetFontNumber` using each `name`, and update the `realID` field with the current font number for that name. If a font isn't present, you should warn the user: You could let the user choose an alternative font, or use the default application font by storing (and using) the constant `applFont` as that font's `realID`; this way, the user could still edit the document, but the original font name would remain, so that when the user adds the proper font to the System file, or moves the document to a Macintosh whose System file contains it, the document would be displayed as originally intended.

The overhead of handling your documents' fonts in this manner is rather small; as more font families become available, and Font/DA Mover's renumberings occur more often, your customers will appreciate this extra effort.

Further Reference:

- The Font Manager