Technical Note TN2024

The Mac OS X Font Manager

CONTENTS

The Font Manager on Mac OS X offers advanced features with a simplicity of design that makes it easy to support an extensive range of font technologies and data formats. The programming interface is designed with performance, scalability, and consistency in mind, and is available to Cocoa and Carbon applications through the Apple Type Services (ATS) framework and Quickdraw framework on Mac OS X.

Updated: [Jul 11 2001]

How are fonts installed?

Installing fonts is a simple process of copying or moving files to any of the standard font directories of the file system on Mac OS X. Note that the directories of the file system are arranged so that resources local to the user's computer are segregated from those on the network, and, on a computer, system resources are segregated from those under the control of the user or system administrator. Applications, documents, fonts, and other resources should go in one of several file-system domains. The domain is an area of the file system segregated from other domains and with structural elements identical to other domains.

Changes to the font directories are registered with the operating system when an application is launched or a user logs in to the account or computer on which the changes occurred. Duplicate fonts are resolved based on the order of precedence defined for the standard domains and are described from highest to lowest priority in Table 1.

~/Library/Fonts (User)

The User domain is specific to the user who is logged into the system and is associated with the user's home directory, which can either be on the boot volume or on the network. The user has complete control over the contents of this domain.

/Library/Fonts (Local)

The Local domain is for fonts shared among all users of a particular computer and not required by the operating system to run. Users with system administrator privileges can add, remove, and modify items in this domain, which is also the recommended location for fonts that are shared among applications.

/Network/Library/Fonts (Network)

The Network domain is for fonts shared among all users of a local area network. The contents of this domain are typically located on network file servers and are under the control of a network administrator.

/System/Library/Fonts (System)

The System domain contains the default fonts required by the operating system to run and should not be altered.

[Classic System Folder]/Fonts (Classic)

The Fonts folder of the user's preferred Classic System folder is the domain with the lowest precedence of all the font directories and is registered with the operating system even when the Classic compatibility environment is not running. Note that for applications running in the Classic compatibility environment the inverse is not true, and none of the other domains besides the Fonts folder in the Classic System folder are available.

Table 1. Font directories and standard domains.

The domain in which a font is placed defines the scope of applicability or accessibility for that font. For example, if a user installs a custom font in the user domain, the font is available only to that user. If an administrator installs the same font in the network domain, the font is available to everyone on the network.

The ownership and permissions model of the file system is fundamentally different on Mac OS X from previous releases of the operating system. This difference affects how you install and use fonts. For each file and directory in the file system there are three categories of users (owner, group, and other), and for each type of user there are three specific permissions that affect access to the file or directory (read, write, and execute). When you install a font, check that the permissions of the files associated with the font are set to enable read access for the appropriate categories of users for the domain.

Back to top

Font technologies and data formats

The font technologies and data formats supported for rendering, print-preview, and printing on Mac OS X are listed in Table 2.

Macintosh TrueType font suitcase
Windows TrueType TTF/TTC outline/bitmapped font
PostScript OpenType Roman outline/bitmapped font
PostScript OpenType CID Chinese/Japanese/Korean/Vietnamese outline/bitmapped font
PostScript Type 1 outline font with Macintosh bitmapped font suitcase (LWFN)
Macintosh PostScript Type 1 enabled font suitcase (SFNT)
Macintosh PostScript Type 1 CID enabled font suitcase (SFNT/CID).

Table 2. Font technologies and data formats.

Note that PostScript Type 1 Multiple Master fonts are not yet supported on Mac OS X.

The contents of the .dfont file format used with all the standard TrueType system fonts on Mac OS X are identical to the standard font suitcase files, except that the font resources are stored in the data fork of the file. On Mac OS X, the `'FONT'` bitmapped font resources are not supported, although the `'NFNT'` bitmapped font resources are supported for Quickdraw applications. Font families consisting entirely of `'NFNT'` bitmapped font resources are ignored by non-Quickdraw applications based on Cocoa, Apple Type Services for Unicode™ Imaging (ATSUI), and Multilingual Text Editor (MLTE), and cannot be applied to user interface elements such as menus, windows, and static and editable text controls.

When support was first provided on Mac OS for Chinese/Japanese/Korean/Vietnamese fonts, it was necessary to introduce a new format for bitmapped fonts to replace the `'FONT'` and `'NFNT'` resource formats. Fonts in the new format were referred to as `'fbit'` fonts, in which basic information on the font continued to reside in the `'FOND'` resource but the bitmapped data resided in an external file. There are no plans to support the `'fbit'` (marukan) packaging format on Mac OS X, and font developers and users must convert to using an alternate bitmapped font format (`'sbit'`) associated with TrueType and OpenType fonts.

Back to top

What programming interfaces are supported?

The programming interface defined by the ATS framework on Mac OS X consolidates the font-related functions designed and implemented for optimal performance and compatibility with the other components of the operating system, including Quartz, Quickdraw, Cocoa, and ATSUI. The functions declared in the interface files of the ATS framework are not available on CarbonLib for CFM based applications. To use these functions from a CFM-based applications you need to link against CarbonFrameworkLib or use the techniques described in the CallMachOFramework sample code from the Carbon SDK.

The extended Font Manager functions first implemented on Mac OS 9.0 are exported by the Quickdraw framework on Mac OS X for dyld-based applications and by the Carbon framework and CarbonLib for CFM based applications. The programming interface provides a migration path to Mac OS X for Carbon applications and is described in the Font Manager developer documentation. The functions are not available on earlier releases of the system software, including Mac OS 8.6 and any version of CarbonLib installed on those system software releases. To check if the extended Font Manager functions are implemented you can verify that the address of any exported symbol from the programming interface has been resolved by the Code Fragment Manager as described in Listing 1.

```
if ( (UInt32)kUnresolvedCFragSymbolAddress != (UInt32)FMGetGeneration )
```

```
... // The extended Font Manager API is implemented.
```

Listing 1. A source code listing to check if the extended Font Manager functions are implemented.

Back to top

What are the differences in the data types?

The base types for `ATSFontFamilyRef` (opaque 32-bit value) and `FMFontFamily` (signed 16-bit integer) are different, so you should avoid type casting or implicit type promotion when working with these data types. Instead, use the conversion functions defined for the font family references by the Quickdraw framework to protect your software from any changes or differences in the way these two data types are generated.

`FMFont` and `ATSUFontID` are equivalent data types that may be used interchangeably with the functions provided by the Font Manager and ATSUI in the Quickdraw framework. You must still use the conversion functions defined for the font references by the Quickdraw framework when handling the `ATSFontRef` data type.

Back to top

Using the data access functions

You can learn more about the information accessible through the functions `FMGetFontTable` and `FMGetFontTableDirectory` from the TrueType specification defined in the TrueType Reference Manual, and the OpenType specification described in the Adobe Type Technology Technical Resources and Microsoft Typography Technical Information web sites.

The function `FMGetFontFormat` returns values misleadingly labeled as font technologies in the interface file Fonts.h, of the Quickdraw framework. The `kFMTrueTypeFontTechnology` (`'true'`) format tag is reserved for TrueType fonts. The `kFMPostScriptFontTechnology` (`'typ1'`) format tag indicates the class of PostScript fonts that consist of a single font resource (`'sfnt'`) that matches the data format used in TrueType and OpenType fonts. The font data is located in the data tables specified and tagged for the particular font technology. An example is the PostScript CID-Keyed SFNT font file format described in the Adobe Font Technical Note 5180. The additional font format tag, `'LWFN'`, represents the traditional packaging format for PostScript Type 1 fonts on Mac OS (i.e., bitmapped font suitcase with one or more auxiliary PostScript outline files).

The function `FMGetFontContainer` provides a reference to the font suitcase rather than the PostScript outline file for `'LWFN'` class fonts. This means that there is no single function implemented to obtain the data stored in the PostScript outline file.

Back to top

Using font names

The PostScript name located in the `'FOND'` resource or the `'name'` table of the TrueType and OpenType fonts is used as a persistent reference to a font by Quartz and other PostScript-based software like that used in LaserWriter printers. The font family, font, and sub-family names located in the `'name'` table define the entries in the font panel provided by Cocoa applications.

The Quickdraw name provided by `GetFontName` (or `FMGetFontFamilyName`) was originally the resource name of the `'FOND'` resource of the font family. It is used in Quickdraw applications as a persistent reference to the font family and also in user interface in the Fonts menu generated from the `AppendResMenu` function. Note that you also need the Quickdraw style to specify a component font in the font family.

Back to top

Using the standard font menus

There is a problem in the standard font menu functions described in the chapter in the Font Manager documentation on "Handling the Standard Font Menu". TrueType variation fonts like Skia and Hoefler Text may not work properly for the hierarchical form of the standard font menu in Quickdraw applications.

Back to top

Is there notification of changes to the font database?

There is no support at this time for notification of changes to the font database, but you can poll for changes to the generation counter as described in the documentation of the Font Manager on the FMGeneration data type.

Back to top

Changes required when working with the Resource Manager

On Mac OS X, the font files are not accessible through a globally shared system resource chain and may not even contain font data in the resource fork of the file. You cannot depend on Resource Manager functions like `GetResource` and `GetResInfo` to access the information stored in the font files. Instead, you should use the general data access functions defined in the ATS and Quickdraw frameworks, which have been designed to work independently of the file formats and technologies of the fonts. The functions defined by the ATS framework that provide information that corresponds most closely to the data stored in the `'FOND'` resource include: `ATSFontFamilyFindFromName`, `ATSFontFamilyGetName`, `ATSFontFamilyGetEncoding`, and `ATSFontGetPostScriptName`. Note that the function `ATSFontGetHorizontalMetrics` does not provide the overall font family metrics located in the `'FOND'` resource and is based instead on the data tables stored in the component fonts.

For additional information on adapting resource-based source code for Mac OS X, refer to the chapter in the Font Manager documentation on "Rewriting Resource-based Code".

The function `ATSFontGetFontFamilyResource` defined in the interface file ATSFont.h, of the ATS framework lets you access the contents of the `'FOND'` resource that references a particular font. The function copies the contents of the `'FOND'` resource to a data buffer that you must allocate prior to using the function. If you call `ATSFontGetFontFamilyResource` with the data buffer parameter set to nil, the size of the buffer required to hold the `'FOND'` resource data will be returned in the buffer size parameter. You can then use this information to allocate the required buffer.

To access other font resources of a Macintosh TrueType font suitcase, including the outline (`'sfnt'`) and bitmapped (`'NFNT'`) font resources, use `FMGetFontContainer` to obtain a file reference that may be passed to the Resource Manager functions to open the file and access the resource data directly. The source code listing in Listing 2 provides basic support for accessing the `'FOND'` resource on Mac OS X and Mac OS 9. Note that the sample code is dependent on the extended Font Manager functions and will not work on earlier releases of the system software that do not support this programming interface.

```
OSStatus GetFontFamilyResource(FMFontFamily iFontFamily, Handle* oHandle) {
    FMFont font;
    Str255 fontFamilyName;
    SInt16 rsrcFRefNum;
    Handle rsrcHandle;
    FSSpec rsrcFSSpec;
    FSRef rsrcFSRef;
    HFSUniStr255 forkName;
    OSStatus status;

    font = kInvalidFont;
    rsrcFRefNum = -1;
    rsrcHandle = NULL;
    status = noErr;


        /* Get the font family name to use with the Resource
        Manager when grabbing the 'FOND' resource. */
    status = FMGetFontFamilyName(iFontFamily, fontFamilyName);
    require(status == noErr, FMGetFontFamilyName_Failed);

        /* Get a component font of the font family to obtain
        the file specification of the container of the font
        family. */
    status = FMGetFontFromFontFamilyInstance(iFontFamily, 0, &font, nil);
    require(status == noErr && font != kInvalidFont,
        FMGetFontFromFontFamilyInstance_Failed);

    status = FMGetFontContainer(font, &rsrcFSSpec);
    require(status == noErr, FMGetFontContainer_Failed);

        /* Open the resource fork of the file. */
    rsrcFRefNum = FSpOpenResFile(&rsrcFSSpec, fsRdPerm);

        /* If the font is based on the ".dfont" file format,
        we need to open the data fork of the file. */
    if ( rsrcFRefNum == -1 ) {
            /* The standard fork name is required to open
            the data fork of the file. */
        status = FSGetDataForkName(&forkName);
        require(status == noErr, FSGetDataForkName_Failed);

            /* The file specification (FSSpec) must be converted
            to a file reference (FSRef) to open the data fork
            of the file. */
        status = FSpMakeFSRef(&rsrcFSSpec, &rsrcFSRef);
        require(status == noErr, FSpMakeFSRef_Failed);

        status = FSOpenResourceFile(&rsrcFSRef,
            forkName.length, forkName.unicode,
            fsRdPerm, &rsrcFRefNum);
        require(status == noErr, FSOpenResourceFile_Failed);
    }

    UseResFile(rsrcFRefNum);

        /* On Mac OS X, the font family identifier may not
        match the resource identifier after resolution of
        conflicting and duplicate fonts. */
    rsrcHandle = Get1NamedResource(FOUR_CHAR_CODE('FOND'), fontFamilyName);
    require_action(rsrcHandle != NULL,
        Get1NamedResource_Failed, status = ResError());

    DetachResource(rsrcHandle);

Get1NamedResource_Failed:

    if ( rsrcFRefNum != -1 )
        CloseResFile(rsrcFRefNum);

FSOpenResourceFile_Failed:
FSpMakeFSRef_Failed:
FSGetDataForkName_Failed:
FMGetFontContainer_Failed:
FMGetFontFromFontFamilyInstance_Failed:
FMGetFontFamilyName_Failed:

    if ( oHandle != NULL )
        *oHandle = rsrcHandle;

    return status;
}
```

Listing 2. A source code listing to access the font information stored in a resource-based font suitcase file.

You cannot assume that converting a font family instance to a font reference using the function `FMGetFontFromFontFamilyInstance`, then applying the inverse operation using the function `FMGetFontFamilyInstanceFromFont` results in the original font family instance. This problem is most apparent when you work with fonts referenced by multiple font families. It can also occur in functions like `ATSFontGetFontFamilyResource` and `FMGetFontContainer` that implicitly convert a font reference to a font family instance. You can adjust for any differences in the font family instance that you obtain by determining its effective style based on the intrinsic style provided by the `FMGetFontFromFontFamilyInstance` conversion function as described in Listing 3.

```

FMGetFontFromFontFamilyInstance(originalFamily, originalStyle,
    &font, &intrinsicStyle);

remainingStyle = originalStyle & ~intrinsicStyle;

FMGetFontFamilyInstanceFromFont(font, &newFamily, &newStyle);
if ( newFamily != originalFamily )
    effectiveStyle = newStyle | remainingStyle;
else
    effectiveStyle = originalStyle;
```

Listing 3. A source code listing to handling fonts referenced by multiple font families.

This listing assumes that the font family selected by `FMGetFontFamilyInstanceFromFont` is consistent with the one selected by the data access functions. If not, you must enumerate the instances of the new font family and look for a strike that references the font to compute the required style adjustments.

On Mac OS X, the Resource Manager does not automatically activate fonts stored in the resource fork of the application file. Use `FMActivateFonts` (in Fonts.h of the Quickdraw framework) or `ATSFontActivateFromFileSpecification` (in ATSFont.h of the ATS framework) with the file specification of the application to activate any fonts stored in the resource fork of that file.

The function `AddResMenu` has been modified to work with all supported data formats and particularly with non-resource fork based fonts, including PostScript OpenType and Windows TrueType fonts.

Back to top

Changes required when working with Quickdraw

On Mac OS X, you can continue using the Font Manager routine, `FMSwapFont`, to access the font information generated by Quickdraw but you will not have access to the fields of any data structure that reference a resource-based data handle. This includes the `fontHandle` field of the `FMOutput` data structure and the `tabFont` and `fHand` fields of the `WidthTable` data structure which are set to `NULL` by the Font Manager.

If the `HIToolbox` framework has not been initialized, functions like `GetSysFont` and `GetAppFont` cannot access script system data for the default system fonts and set the return value to zero. The solution is to reference a function in your application that is exported by the `HIToolbox`, like `GetApplicationTextEncoding`.

Downloadables

	Acrobat version of this Note (56K).	Download