Font Management API Preliminary Documentation

For Carbon 1.0 and Mac OS 9

Preliminary Draft

Technical Publications

IMPORTANT

INTRODUCTION

Font management consists of two tasks: installing, activating, and deactivating font files, and maintaining a repository of font data. Traditionally the responsibility for managing the font files on Mac OS has resided with the Finder and Fonts folder, which, in turn, depended on specialized code in the Font Manager and Resource Manager to populate and manage the font database. However, this basic font management system has been limited by an implementation that relies heavily on the Resource Manager, which was not designed to handle the sizeable data storage and retrieval needs of most professional-level font collections. Because of this, third-party font management utilities have been created to address many of the limitations inherent in the System suitcase model. However, in order to provide these services, the font management utilities have had to directly manipulate system font data, a practice that is not supported under Carbon.

To address these issues, beginning with Mac OS 8.5, Apple Type Services for Unicode Imaging ("ATSUI") introduced a text rendering engine and font management model designed from the beginning to handle multiple font technologies and file formats. Additionally, the Font Manager and QuickDraw Text were completely rewritten to integrate the glyph data cache and font file and data management system as an adjunct to the System suitcase model of the Resource Manager. The result is that developers no longer have to depend upon the Resource Manager if they wish to access and modify the font collection. Instead, Carbon 1.0 provides a completely new Font Management API as part of the Font Manager.

Note: While this document discusses the new Font Management functionality in terms of Carbon, these new features of the Font Manager are also available independently of Carbon on Mac OS 9.

Carbon 1.0 provides a transition period for developers to begin using the font access and data management programming interface while still maintaining compatibility with the existing font management model. For versions of Carbon after 1.0, the goal is to complete the transition to the font access and data management programming interface and to entirely eliminate the use of Resource Manager functions to access font information and data.

The Font Manager's font access and data management programming interface supplied in Carbon 1.0 provides the following features.

- Enumerating fonts and font families.
- Accessing information on font families, including name, character encoding, and member fonts.
- Accessing information on fonts, including technology, format, name, foundry, and version.
- Accessing font data.
- Creating and managing a basic font menu.

- Activating and deactivating fonts.
- Notifying clients of changes in the font database.

RELATED INFORMATION

There are additional sources of information for the concepts referenced in this document. A basic understanding of the Resource Manager from *Inside Macintosh: More Macintosh Toolbox*, and QuickDraw Text and the Font Manager from *Inside Macintosh: Text* is assumed throughout the text of this discussion. The design and implementation of the text rendering and font management facilities of ATS Unicode are described in *Inside Macintosh: Rendering Text with Apple Type Services for Unicode Imaging*.

APPLICATION PROGRAMMING INTERFACE

The application programming interface includes routines for enumerating the fonts and font families stored in the font database, activating and deactivating fonts, and accessing information on the fonts and font families, including the basic name of a font family.

DATA STRUCTURES

This section describes the data structures used in the font access and data management programming interface.

Font Families and Fonts

The font family reference (FMFontFamily) represents a collection of fonts with the same design characteristics. It replaces the standard QuickDraw font identifier and may be used with all QuickDraw functions including GetFontName and TextFont. Unlike the standard QuickDraw font identifier, the font family reference cannot be passed to the Resource Manager to access information from a FOND resource handle. A font family reference does not imply a particular script system, nor is the character encoding of a font family determined by an arithmetic mapping of its value.

```
typedef SInt16                          FMFontFamily;
typedef SInt16                          FMFontStyle;
typedef SInt16                          FMFontSize;
```

The font family is a collection of fonts, each of which is identified by a reference (FMFont) that maps to a single object registered with the font database. The font references associated with the font family consist of individual outline and bitmapped fonts that may be used with the font access routines of the Font Manager and ATS Unicode.

```
typedef UInt32                          FMFont;
```

Each font object maps to one or more combinations of a font family reference and standard QuickDraw style. This is roughly equivalent to the information stored in the font association table of the FOND resource handle, with the exception of a point size descriptor. Since each object represents the entire array of point sizes for a given font, only the font family reference and style are required to fully specify any given font reference.

```
struct FMFontFamilyInstance {
    FMFontFamily                  fontFamily;
    FMFontStyle                   fontStyle;
};
typedef struct FMFontFamilyInstance     FMFontFamilyInstance;
```

Generations

The first release of the font access and data management programming interface provides a simple notification mechanism for changes to the font database. Any operation that adds, deletes, or modifies one or more font family or font references triggers an update of a global generation seed value. Each font family and font reference modified during the transaction is tagged with a copy of the generation seed. The programming interface provides accessor functions for the current value of the generation seed and the generation tag associated with each font family and font reference, which may be used in conjunction with the enumeration functions to identify any changes in the state of the font database.

```
typedef UInt32                          FMGeneration;
```

Filters

You may use the filter data structure to restrict the scope of the enumeration and activation functions to the font families and fonts that match a particular technology, font container, or generation tag.

```
enum {
    kFMCurrentFilterFormat               = 0L
};

typedef UInt32 FMFilterSelector;
enum {
    kFMInvalidFilterSelector             = 0L,
    kFMFontTechnologyFilterSelector      = 1L,
    kFMFontContainerFilterSelector       = 2L,
    kFMGenerationFilterSelector          = 3L,
    kFMFontFamilyCallbackFilterSelector  = 4L,
    kFMFontCallbackFilterSelector        = 5L
};

enum {
    kFMTrueTypeFontTechnology            = FOUR_CHAR_CODE('true'),
    kFMPostScriptFontTechnology          = FOUR_CHAR_CODE('typ1')
};

struct FMFilter {
    UInt32                        format;
    FMFilterSelector              selector;
    union {
        FourCharCode                  fontTechnologyFilter;
        FSSpec                        fontContainerFilter;
        FMGeneration                  generationFilter;
        FMFontFamilyCallbackFilterUPP     fontFamilyCallbackFilter;
        FMFontCallbackFilterUPP       fontCallbackFilter;
    }                             filter;
};
typedef struct FMFilter                  FMFilter;
```

The custom filter function options may be used to restrict the scope of the enumeration and activation functions to an arbitrary set of criteria defined by the client. If any object fails to match the criteria specified in the function the error kFMIterationFilterFailed must be returned.

```
typedef pascal OSStatus (* FMFontFamilyCallbackFilterProcPtr)
                              (FMFontFamily          iFontFamily,
                               void *                iRefCon);
typedef pascal OSStatus (* FMFontCallbackFilterProcPtr)
                              (FMFont                iFont,
                               void *                iRefCon);
```

iFontFamily	Font family reference.
iFont	Font reference.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation.

Option Flags

The OptionBits parameter is a set of flags used to control the behavior specific to a function. Where no options are yet defined, you may pass a neutral value of kFMDefaultOptions for future compatibility. This design makes it possible for a single function to provide a set of related services or variants rather than cluttering the name space with a group of closely related functions. It also reduces the complexity of the functions by reserving a single parameter for all the possible options.

```
typedef UInt32 OptionsBits;
```

```
enum {
    kFMDefaultOptions                  = kNilOptions,
    kFMUseGlobalScopeOption            = 0x00000001
};
```

ENUMERATION

The following routines implement iteration and searching of font families and fonts using a cursor model that is thread-safe and compatible with the multitasking, preemptive environment of Mac OS X. Four routines are provided: one to create an iterator object which maintains the state necessary to provide high quality data, one to iterate over the items, one to reset the state of the iterator, and one to dispose of the iterator when done.

FMCreateFontFamilyIterator

Creates an iterator to access the font family references in the context of the current application process or kernel task.

```
typedef struct FMFontFamilyIterator       FMFontFamilyIterator;

OSStatus
FMCreateFontFamilyIterator     (const FMFilter *      iFilter,
                                void *                iRefCon,
                                OptionBits            iOptions,
                                FMFontFamilyIterator *    ioIterator);
```

iFilter	Filter specification; set this parameter to nil to access all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.
ioIterator	Iterator reference. The data structure that defines the iterator reference should be considered opaque and must not be modified directly by the client.

Discussion: The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. The operating system handles the memory allocation and maintains the state of every iterator. When you have finished using an iterator, you should call the dispose function to free any allocated memory. If any object in the scope is modified, the iterator is invalidated and will return the error kFMIteratorScopeModified until it has been explicitly fixed (by the reset function) or disposed.

FMDisposeFontFamilyIterator

Disposes of a font family iterator.

```
OSStatus
FMDisposeFontFamilyIterator    (FMFontFamilyIterator *    ioIterator);
```

ioIterator Iterator reference.

Discussion: The operating system disposes of the iterator and releases any allocated memory. Further attempts to use the iterator will result in an error.

FMResetFontFamilyIterator

Resets the options and positional state of a font family iterator.

```
OSStatus
FMResetFontFamilyIterator      (const FMFilter *      iFilter,
                                void *                iRefCon,
                                OptionBits            iOptions,
                                FMFontFamilyIterator *    ioIterator);
```

iFilter	Filter specification; set this parameter to nil to access all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.
ioIterator	Iterator reference.

Discussion: The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. You would use this call to completely restart iteration, ignoring any state about objects previously returned.

FMGetNextFontFamily

Accesses the next font family reference in the context of the current application process or kernel task.

```
typedef struct FMFontIterator            FMFontIterator;

OSStatus
FMGetNextFontFamily            (FMFontFamilyIterator *    ioIterator,
                                FMFontFamily *        oFontFamily);
```

ioIterator	Iterator reference.
oFont	Font reference.

Discussion: When the iterator has completed its coverage of the registry, it returns the status code, kFMIterationCompleted.

FMCreateFontIterator

Creates an iterator to access the font references in the context of the current application process or kernel task.

```
OSStatus
FMCreateFontIterator           (const FMFilter *      iFilter,
                                void *                iRefCon,
                                OptionBits            iOptions,
                                FMFontIterator *      ioIterator);
```

iFilter	Filter specification; set this parameter to nil to access all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.
ioIterator	Iterator reference. The data structure that defines the iterator reference should be considered opaque and must not be modified directly by the client.

Discussion: The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. The operating system handles the memory allocation and maintains the state of every iterator. When you have finished using an iterator, you should call the dispose function to free any allocated memory. If any object in the scope is modified, the iterator is invalidated and will return the error kFMIteratorScopeModified until it has been explicitly fixed (by the reset function) or disposed.

Example:
```
FMFilter            filter;
FMFontIterator      iterator;
...
filter.format = kFMCurrentFilterFormat;
filter.selector = kFMFontTechnologyFilterSelector;
filter.fontTechnologyFilter = kFMTrueTypeFontTechnology;
status = FMCreateFontIterator(&filter, NULL, kFMDefaultOptions, &iterator);
```

FMDisposeFontIterator

Disposes of a font iterator.

```
OSStatus
FMDisposeFontIterator          (FMFontIterator *      ioIterator);
```

ioIterator Iterator reference.

Discussion: The operating system disposes of the iterator and releases any allocated memory. Further attempts to use the iterator will result in an error.

FMResetFontIterator

Resets the options and positional state of a font iterator.

```
OSStatus
FMResetFontIterator            (const FMFilter *      iFilter,
                                void *                iRefCon,
                                OptionBits            iOptions,
                                FMFontIterator *      ioIterator);
```

iFilter	Filter specification; set this parameter to nil to access all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.
ioIterator	Iterator reference.

Discussion: The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. You would use this call to completely restart iteration, ignoring any state about objects previously returned.

FMGetNextFont

Accesses the next font reference in the context of the current application process or kernel task.

```
OSStatus
FMGetNextFont                  (FMFontIterator *      ioIterator,
                                FMFont *              oFont);
```

ioIterator Iterator reference.

oFont	Font reference.
Discussion:	When the iterator has completed its coverage of the registry, it returns the status code, kFMIterationCompleted.

FONT FAMILIES

The font family reference (FMFontFamily) is a 16-bit font family identifier compatible with the standard Macintosh Toolbox routines of the Font Manager, QuickDraw Text, Resource Manager, and Script Manager. It may be used wherever the standard QuickDraw font identifier is used, including the txFont field of the QuickDraw graphics port and TextEdit data structures.

FMCreateFontFamilyInstanceIterator

Creates an iterator to access the components of a font family.

```
typedef struct FMFontFamilyInstanceIterator   FMFontFamilyInstanceIterator;

OSStatus
FMCreateFontFamilyInstanceIterator
                                (FMFontFamily      iFontFamily,
                                 FMFontFamilyInstanceIterator * ioIterator);
```

iFontFamily	Font family reference.
ioIterator	Iterator reference. The data structure that defines the iterator reference should be considered opaque and must not be modified directly by the client.
Discussion:	The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. The operating system handles the memory allocation and maintains the state of every iterator. When you have finished using an iterator, you should call the dispose function to free any allocated memory. If any object in the scope is modified, the iterator is invalidated and will return the error kFMIteratorScopeModified until it has been explicitly fixed (by the reset function) or disposed.

FMDisposeFontFamilyInstanceIterator

Disposes of a font family instance iterator.

```
OSStatus
FMDisposeFontFamilyInstanceIterator
                                (FMFontFamilyInstanceIterator * ioIterator);
```

ioIterator	Iterator reference.
Discussion:	The operating system disposes of the iterator and releases any allocated memory. Further attempts to use the iterator will result in an error.

FMResetFontFamilyInstanceIterator

Resets the options and positional state of a font family instance iterator.

```
OSStatus
FMResetFontFamilyInstanceIterator
                                (FMFontFamily      iFontFamily,
                                 FMFontFamilyInstanceIterator * ioIterator);
```

iFontFamily	Font family reference.

ioIterator	Iterator reference. The data structure that defines the iterator reference should be considered opaque and must not be modified directly by the client.

Discussion:	The iterator is initialized to the "start of iteration" state, meaning that it is not positioned on any object and all objects in the registry have yet to be returned. You would use this call to completely restart iteration, ignoring any state about objects previously returned.

FMGetNextFontFamilyInstance

Accesses the next component of a font family.

```
OSStatus
FMGetNextFontFamilyInstance    (FMFontFamilyInstanceIterator * ioIterator,
                                FMFont *              oFont,
                                FMFontStyle *         oStyle,
                                FMFontSize *          oSize);
```

ioIterator	Iterator reference.
oFont	Font reference.
oStyle	Intrinsic style of font.
oSize	Intrinsic size of font.

Discussion:	When the iterator has completed its coverage of the font family, it returns the status codekFMIterationCompleted.

FMGetFontFamilyFromName

Finds the font family associated with a standard QuickDraw name.

```
FMFontFamily
FMGetFontFamilyFromName        (ConstStr255Param      iName);
```

name	QuickDraw font family name.
function result	Font family reference.

Discussion:	If the font family specified by the name parameter does not exist, the function returns a value of kInvalidFontFamily. This function is equivalent to the GetFNum function without the compatibility checks for shadowing the FOND resource chain.

Example: `TextFont(FMGetFontFamilyFromName("\pCharcoal"));`

FMGetFontFamilyName

Obtains the standard QuickDraw name of a font family.

```
OSStatus
FMGetFontFamilyName            (FMFontFamily          iFontFamily,
                                Str255                oName);
```

iFontFamily	Font family reference.
oName	QuickDraw font family name.

Discussion:	If the font family specified by the iFontFamily parameter does not exist, the function returns a name value set to an empty string. This function is equivalent to the GetFontName function without the compatibility checks for shadowing the FOND resource chain.

FMGetFontFamilyTextEncoding

Obtains the text encoding of a font family.

```
OSStatus
FMGetFontFamilyTextEncoding    (FMFontFamily          iFontFamily,
                                TextEncoding *        oTextEncoding);
```

iFontFamily	Font family reference.
oTextEncoding	Text encoding.

Discussion:	Unlike the FontToScript function of the Script Manager, the state of the font force flag is ignored and the script system of the font family is not mapped to the zero even if it is disabled in the current application process or kernel task.

FMGetFontFamilyGeneration

Obtains the generation of a font family.

```
OSStatus
FMGetFontFamilyGeneration      (FMFontFamily          iFontFamily,
                                FMGeneration *        oGeneration);
```

iFontFamily	Font family reference.
oGeneration	Generation tag.

FONTS

FMGetFontFormat

Retrieves the format identifier of a font.

```
OSStatus
FMGetFontFormat                (FMFont                iFont,
                                FourCharCode *        oFormat);
```

iFont	Font reference.
oFormat	Format tag.

FMGetFontTableDirectory

Retrieves a copy of the table directory of a font.

```
OSStatus
FMGetFontTableDirectory        (FMFont                iFont,
                                ByteCount             iLength,
                                void *                iBuffer,
                                ByteCount *           oActualLength);
```

iFont	Font reference.
iLength	Size of the data buffer allocated by the client.
iBuffer	Data buffer for the font table directory; set this parameter to nil to obtain only the length of the table directory, not its contents.
oActualLength	Length of font table directory.

Discussion: If an error occurs and the pointer cannot be obtained, the function returns a table directory value set to nil.

FMGetFontTable

Retrieve a copy of all or part of a data table for a font; the data table must be identified by its tag identifier in the table directory.

```
OSStatus
FMGetFontTable                (FMFont             iFont,
                               FourCharCode       iTag,
                               ByteOffset         iOffset,
                               ByteCount          iLength,
                               void *             iBuffer,
                               ByteCount *        oActualLength);
```

iFont	Font reference.
iTag	Font table tag identifier.
iOffset	Offset to partial font table data.
iLength	Size of the data buffer allocated by the client.
iBuffer	Data buffer for the font table directory; set this parameter to nil to obtain only the length of the table directory, not its contents.
oActualLength	Length of font table directory.

Discussion: The offset is relative to the beginning of the data table and is zero-based; the length is the number of bytes to be retrieved starting at the offset. If an error occurs and the data cannot be obtained, including an inconsistent set of values for the offset and length, the function returns a buffer value set to nil.

FMGetFontGeneration

Obtains the generation of a font.

```
OSStatus
FMGetFontGeneration           (FMFont             iFont,
                               FMGeneration *     oGeneration);
```

iFont	Font reference.
oGeneration	Generation tag.

FMGetFontContainer

Obtains the font container of a font.

```
OSStatus
FMGetFontContainer            (FMFont             iFont,
                               FSSpec *           oFontContainer);
```

iFont	Font reference.
oFontContainer	File specification to the font container.

CONVERSION

FMGetFontFromFontFamilyInstance

Finds the font reference associated with a standard QuickDraw style in the context of a font family.

```
OSStatus
FMGetFontFromFontFamilyInstance
                                (FMFontFamily          iFontFamily,
                                 FMFontStyle           iStyle,
                                 FMFont *              oFont,
                                 FMFontStyle *         oIntrinsicStyle);
```

iFontFamily	Font family reference.
iStyle	Style.
oFont	Font reference.
oIntrinsicStyle	Intrinsic style of font.

Discussion: If the font specified by the font family and style fields of the font family instance does not exist, the function determines which of several mismatched QuickDraw styles is the closest match, otherwise it returns a value of kInvalidFont.

FMGetFontFamilyInstanceFromFont

Finds the font family and standard QuickDraw style associated with a font.

```
OSStatus
FMGetFontFamilyInstanceFromFont
                                (FMFont                iFont,
                                 FMFontFamily *        oFontFamily,
                                 FMFontStyle *         oStyle);
```

iFont	Font reference.
oFontFamily	Font family.
oStyle	Style.

ACTIVATION

Fonts are activated and deactivated in groups defined by their representation in the file system as font suitcase files or other font file formats supported by the font access and data management programming interface. In Carbon 1.0, the fonts must be referenced by a standard QuickDraw font family resource in the resource fork of a file, or stored as standard TrueType fonts or PostScript font programs in the data fork of a file.

FMActivateFonts

Activates one or more fonts in a font container.

```
OSStatus
FMActivateFonts                 (const FSSpec *        iFontContainer,
                                 const FMFilter *      iFilter,
                                 void *                iRefCon,
                                 OptionBits            iOptions);
```

iFontContainer	Font container.
iFilter	Filter specification; set this parameter to nil to activate all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.

Discussion: This function activates one or more fonts stored in a file. The filter parameter may be used to restrict the scope of the function to a particular font or fonts in the font container.

FMDeactivateFonts

Deactivates one or more fonts in a font container.

```
OSStatus
FMDeactivateFonts              (const FSSpec *        iFontContainer,
                                const FMFilter *      iFilter,
                                void *                iRefCon,
                                OptionBits            iOptions);
```

iFontContainer	Font container.
iFilter	Filter specification; set this parameter to nil to deactivate all objects in the scope of the operation.
iRefCon	Reference constant passed to a custom filter function for each object accessed by the operation. If a custom filter function is not defined, use nil as the value of this parameter.
iOptions	Option flags; use kFMDefaultOptions for the default behavior. The option flag, kFMUseGlobalScopeOption, controls the scope of the operation, whether it is applied to all clients on the system or only in the current context.

FMGetGeneration

Retrieves the generation seed.

```
FMGeneration
FMGetGeneration                    (void);
```

function result Generation seed.

Discussion: Any operation that adds, deletes, or modifies one or more font family or font references triggers an update of a global generation seed value. This function retrieves the current value of the generation seed, which may be used in conjunction with the enumeration functions to identify any changes in the state of the font database.

Example:
```
FMFilter           filter;
FMFontIterator     iterator;
...
filter.format = kFMCurrentFilterFormat;
filter.selector = kFMGenerationFilterSelector;
filter.generationFilter = FMGetGeneration();
status = FMCreateFontIterator(&filter, NULL, kFMDefaultOptions, &iterator);
```

Example:
```
if ( gStandardFontMenuData.seed != FMGetGeneration() )
{
    for ( index = 1; index <= gStandardFontMenuData.itemCount; index++ )
        DeleteMenuItem(theMenu, gStandardFontMenuData.firstItem);
```

```
                status = CreateStandardFontMenu(theMenu,
                                                 gStandardFontMenuData.firstItem - 1,
                                                 gStandardFontMenuData.firstHierMenuID,
                                                 gStandardFontMenuData.options,
                                                 &gHierMenuCount);
            }
```

USER INTERFACE

The Menu Manager provides the application with a standard user interface for soliciting font choices from the user.

CreateStandardFontMenu

Creates a standard font menu.

```
pascal OSStatus
CreateStandardFontMenu         (MenuRef              menu,
                                MenuItemIndex        afterItem,
                                MenuID               firstHierMenuID,
                                OptionBits           options,
                                ItemCount *          outHierMenuCount);
```

menu	The handle to the menu structure to which you wish to add the standard font menu items. The menu handle must be allocated first using NewMenu or GetMenu.
afterItem	The item number of the menu item after which the new menu items are to be added. Specify 0 in the afterItem parameter to insert the new items before the first menu item; specify the item number of a current menu item to insert the new menu items after it; specify a number greater than or equal to the last item in the menu to append the new items to the end of the menu.
firstHierMenuID	The first identifier to use if any hierarchical menus are created.
options	Option flags; use kNilOptions for the default behavior. Use the option flag kHierarchicalFontMenuOption to construct a hierarchical font menu.
outHierMenuCount	The number of hierarchical menus attached to the standard font menu.

Discussion: An application should use this function--instead of specifying the resource type ‘FONT’ or ‘FOND’ and calling the AppendResMenu or InsertResMenu function--to add the names of all the objects in the font database as menu items in its Font menu.

Example:

```
MenuRef     theMenu;
...
theMenu = GetMenuHandle(kFontMenuResID);
status = CreateStandardFontMenu(theMenu,
                                CountMenuItems(theMenu),
                                kFontMenuFirstHierMenuID,
                                kHierarchicalFontMenuOption,
                                &gHierMenuCount);
```

UpdateStandardFontMenu

Updates a standard font menu.

```
pascal OSStatus
UpdateStandardFontMenu         (MenuRef              menu,
                                ItemCount *          outHierMenuCount);
```

menu	Menu handle.
outHierMenuCount	The number of hierarchical menus attached to the standard font menu.

Example:
```
SInt16      thePart;
MenuRef     theMenu;
...
thePart = FindWindow(theEvent ->where, &theWindow);
if ( thePart == inMenuBar )
{
    theMenu = GetMenuHandle(kFontMenuResID);
    status = UpdateStandardFontMenu(theMenu, gHierMenuCount);
    if ( status == noErr )
        status = DoMenuCommand(MenuSelect(theEvent->where));
}
```

GetFontFamilyFromMenuSelection

Finds the font family reference and style from the menu identifier and menu item number returned by MenuSelect.

```
pascal OSStatus
GetFontFamilyFromMenuSelection(MenuRef               menu,
                               MenuItemIndex         item,
                               FMFontFamily *        outFontFamily,
                               FMFontStyle *         outStyle);
```

menu	Menu handle.
item	Menu item index.
outFontFamily	Font family reference of selected item.
outStyle	Style of selected item.

Example:
```
MenuRef         menu;
FMFontFamily    fontFamily;
FMFontStyle     style;
...
menuResult = MenuSelect(theEvent->where);
menuID = HiWord(menuResult);
menuItem = LoWord(menuResult);
if ( menuID == kFontMenuResID ||
   ( menuID >= kFontMenuFirstHierMenuID && menuID <= kFontMenuFirstHierMenuID + gHierMenuCount ))
{
    menu = GetMenuHandle(menuID);
    status = GetFontFamilyFromMenuSelection(menu, menuItem, &fontFamily, &style);
    if ( status == noErr )
    {
        TextFont(fontFamily);
        TextFace(style);
    }
}
```