FontSync API Preliminary Documentation

For FontSync 1.0

Preliminary Draft

Technical Publications

FontSync API Preliminary Documentation

Important

0.0 Introduction

This document describes the features delivered in FontSync 1.0 from an external engineering perspective. The deliverables include a new API and associated shared library and a control panel. This document describes the API and control panel functionality.

1.0 Overview of the Problem

FontSync is a self-contained subsystem intended to address font management issues of great importance among Apple’s high-end publishing customers.

Our publishing customers have long had to endure a host of font-related problems when using Mac OS systems. It is a tribute to their patience and inventiveness that they’ve been so successful at coping with these problems, but their complaints have been heard loud and clear at Customer Advisory Board meetings, and this proposal is a first step at alleviating them.

What are some of the problems faced by printing customers in using fonts?

Prepress customers face many font-related issues, including dealing with huge numbers of fonts and the problems the Mac OS has with a large font folder. However, the number one problem such customers face is output errors caused by font mismatches. The content creator sees one thing on their screen and proof output, sends the document for prepress, and the prepress company gets different results because the same font isn’t used or the font doesn’t work when RIPed.

Some of the reasons that fonts don’t match include:

- Use of TrueType fonts. There is nothing intrinsically wrong with TrueType, but most high-end RIPs cannot render them properly (a problem with the RIP, not Mac OS or applications). Also, there is no way to distinguish between the TrueType and Type 1 versions of a font if they have the same name (e.g., “Times”). Though they have the same name, the metrics are often different, causing reflow. Installers may replace one version with another.

- Different versions of fonts. Even if both the content creator and the prepress company use the same type technology, and think they are using the same font, the fonts may not contain the same data. Metrics, kerning, and appearance can differ, causing reflows. Fonts can be corrupted or changed by users.
- Silent failures to access the font. If the LaserWriter driver cannot access the requested font (for example, if the limit on the number of open files in Mac OS is exceeded) it will silently substitute a bitmap version. This results in expensive, wasted output. This can also happen for other reasons.

Most prepress customers deal with these issues every single day. Comments like "We spend $10-$12 a page dealing with font problems" are typical. The worst example I have heard so far occurred in an issue of the magazine "Woman's Day." One of the designers at the magazine got tired of manually tweaking kerning in Quark XPress, and changed the kerning table in their fonts so that it happened automatically. Of course, when the magazine was sent to prepress, the prepress company had the unmodified version of the font. As a result, the text reflowed, and the last line of every article in the magazine was cut off. The error wasn't caught in proofs, and in fact wasn't caught until the magazine was on the press and many of the subscriptions had already been mailed. An employee at the printing plant pulled a copy to read, and noticed the last line of the article she was reading wasn't there. The mailed copies had to be recalled and the press run redone, and the prepress company had to eat the total cost, which was about $1 million.

This is an extreme example, but prepress customers at Apple Customer Advisory Boards report dropping hundreds of thousands of dollars on similar problems routinely. They are eager for Apple to solve this problem and say it would be a major reason to stay with Mac OS instead of migrating to Windows.

2.0 FontSync Concepts

To solve problems like the ones just described, a mechanism needs to exist to guarantee that the fonts the user uses in a document on one machine are exactly the same as the fonts that will be used on another (or meet less stringent, user-defined criteria for matching). FontSync is an API which applications can adopt in order to solve such customer font problems.

FontSync comprises several parts:

- FontSync Font References, which are an unambiguous way of referring to a specific font, including what kind it is (TrueType, Type 1, etc.), which version it is, checksums of the data, and so on. These FontSync font references can be stored in documents, sent over a network, and turned into references to actual fonts on a host system.
- FontSync Profiles, which are collections of FontSync font references. These profiles can be generated by print shops and given to users to control the set of fonts the user may use.
- An API for iterating, identifying and matching fonts on a host system. This API is used for constructing and matching FontSync font references. It is based on and extends the font matching proposal which is part of the CSS2 Web specification. This API is also used to convert between FontSync font references and actual fonts on a host, of any kind (TrueType, Type 1, bitmap, OpenType, and so on).
- An API for applications to build font menus and other font selection human interface parts. Using a FontSync profile (described above), an application could build a menu containing only the fonts a particular prepress company supports.

In addition to the parts described here, there are other facets which, while not part of FontSync itself, are nevertheless needed for the FontSync proposal to be able to solve real problems. These include:

- Printer driver changes. On the Mac OS, the LaserWriter driver and Download Manager will have to be modified to embed FontSync font references into a PostScript stream and interpret them as they come by.
- Evangelizing application developers to adopt the FontSync model. We have a strong commitment on the part of the major publishing customers to heavily evangelize the application developers to adopt FontSync.
- Working with other font technology houses on getting FontSync accepted as a standard.

2.1 FontSync Font References

A FontSync font reference is an unambiguous way of specifying a particular font. It contains enough information to precisely describe a font in as little space as possible. To do this, it uses data in the font itself (names from the 'name' table, for instance), checksums of the font's pieces (the 'FOND' and 'sfnt' or 'LWFN' resources, for instance). The fact that the data in a reference is algorithmically derived from the font data is what distinguishes FontSync from the approach taken in PANOSE specifications: PANOSE numbers are subjectively assigned by the font designer.

Font references are always derived from a particular, concrete font, and contain all the characterizing information which FontSync knows about. Inexact font comparison is handled by optionally ignoring some of this information at the matching stage.

The internals of a FontSync font reference are private to the implementation; accessor functions as well as constructors and destructors are part of the API. A typical version 1.0 FontSync font reference will be about 2K – 3K bytes long.

2.2 FontSync Profiles

A FontSync profile is a file that is basically a collection of FontSync font references. The profile defines a set of fonts. It is, however not organized as a simple array of FontSync font references. Instead, the data is "inverted" to improve the efficiency of matching against the FontSync Profile. The details of the file structure are private to the implementation.

FontSync profiles are never mandatory; applications are permitted to deal only with FontSync font references, if they so choose.

2.3 Matching Fidelity

FontSync specifies the degree of matching required via a bit mask which specifies which aspects of the font need to match. Among the aspects which might be selected are:

Metric data

Glyph outline data

"Advanced layout" data (e.g., AAT tables)

Font names (from name table)

Postscript data (font Postscript name and glyph Postscript names)

Font appearance (e.g., Panose?)

3.0 Goals and Requirements

It is possible to do some rather sophisticated things when matching fonts. For example, a newer version of a font often contains more glyphs than the previous one. However, if the glyphs carried over from the previous version of the font have not changed (outlines and metrics are identical), it would be reasonable to consider the newer font as matching the older one. One can easily imagine even more interesting font substitutions that could be performed by FontSync. For the customers we are initially targeting, however, the primary requirement is that fonts not be misidentified. Sophisticated font substitution is not a requirement at this stage. To put it another way, false negatives are OK, but false positives are anathema.

Another consideration is the size of a FontSync font reference. To do the sort of detailed matching described in the preceding paragraph, a font reference will have to carry more and more data along. Eventually, it starts approaching the size of a font. At that point, one may as well send the font along with the document and not bother with FontSync references. We must try to keep references as small as possible while still providing a high confidence in the match results. Note that without carrying the original font data instead of a reduced form of it, it is impossible to guarantee that a font reference really does match a given font when a match is reported. The best we can hope for is a very low probability of a false positive.

In FontSync 1.0 we will consider any difference in any particular piece of font data to trigger a mismatch. This will let us use keep a font reference fairly small while still maintaining a high degree of confidence in any match. There will be some spurious mismatches, for instance when glyphs simply got renumbered in a font, but we will not attempt to detect such changes. Matching fonts of different technologies is a particular case of this: it is not a goal of version 1.0. We must be careful, though, to define an API and select data structures that will not preclude adding such sophistication in later versions of FontSync.

A font reference should contain enough information in it so that it can be described to a user in such a way that it can be related to a font the user may have. In particular, it should be possible to recover a (QuickDraw) font family name and some ATSUI-visible font name from a font reference.

FontSync 1.0 must work on Mac OS 9 or later and does not need to run on 68K hardware. Moreover, it must not run on earlier versions of Mac OS. It will eventually be introduced into Mac OS X, so any platform dependencies must be kept isolated as much as possible to minimize the porting effort. It must be usable both by traditional QuickDraw Text-using applications as well as newer ATSUI-based ones.

There needs to be a way for users to create FontSync profiles and to compare them to their existing fonts. There must be a way to set match criteria in the absence of app-specific user interfaces for doing so. We leave the provision of standard canned UI elements to later versions of FontSync.

It must be possible to drive FontSync from AppleScript. This will pose some difficulties because runtime font IDs—font family IDs and font object numbers—are context-dependent. An ID that is valid in the context that receives an Apple Event may not be valid in the context that sent it.

FontSync 1.0 will only work with active fonts. In addition, font references can only be created for fonts that OFA and the font objects system can interpret. That is, fonts for which there is a scaler. This is mainly due to implementation considerations and schedule constraints. Future versions of FontSync can provide a way to do this if required.

Third-party font management utilities can fill in some of the gaps in the total solution to the font problem which FontSync doesn't cover. A few are even starting to provide similar functionality. For example, when no matching active fonts are found, it would be very nice to query a font-management utility to see if it knows of any matching fonts, which it could then auto-activate. One could even imagine such a utility downloading the font from a remote server. FontSync 1.0 will support an Apple Event-based hook for such utilities. This may need to change for Mac OS X, but all of the developers we’ve talked to so far didn’t see that as a big problem.

4.0 Deliverables

FontSync 1.0 will consist of the following components: A shared library that implements the API described in §5, a faceless background application that implements the scripting support described in §7 and a control panel as described in §6. The shared library and faceless background application will be packaged into a single ‘appe’ file, so it will also need an ‘INIT’ resource that registers the shared library with CFM. This file will also contain bundle information for FontSync profiles, including the icons for them. This application should not, however, be launched when a user double-clicks on a profile.

In addition, a set of illustrative sample Apple Scripts will be provided: one that creates a profile, one that compares a profile either to another profile or the the active fonts, and one that changes some of the CP settings.

5.0 FontSync API

The API description uses the C-like syntax of the '.i' files used by the Interfacer tool.

5.1 Types and Constants

5.1.1 Object Types

FontSync font references and profiles are represented by pointers to opaque objects.

```
struct opaque FNSFontReference;
struct opaque FNSFontProfile;
```

5.1.2 Matching Options

Matching options are specified by a bit mask. A mask with all bits off directs FontSync to use a global default option mask (see below).

```
enum unsigned long FNSMatchOptions {
      kFNSMatchNames =              0x00000001,
      kFNSMatchTechnology =         0x00000002,
      kFNSMatchGlyphs =             0x00000004,
      kFNSMatchEncodings =          0x00000008,
      kFNSMatchQDMetrics =          0x00000010,
      kFNSMatchATSUMetrics =        0x00000020,
      kFNSMatchKerning =            0x00000040,
      kFNSMatchWSLayout =           0x00000080,
      kFNSMatchAATLayout =          0x00000100,
      kFNSMatchPrintEncoding =      0x00000200,

      kFNSMissingDataNoMatch =      0x80000000,

      kFNSMatchAll =                0xFFFFFFFF,
      kFNSMatchDefaults =           0
};
```

kFNSMatchNames

All significant names recorded in the font reference must match. This incudes the QuickDraw Text family name, unique name, full name, foundry, manufacturer, version and others. Note that the PostScript names are also examined as part of the `kFNSMatchPrintEncoding` option.

kFNSMatchTechnology

Scaler technologies must match. In principle, it is possible to match other parts of the font across different technologies, but this is not a goal of FontSync 1.0, so even if `kFNSMatchTechnology` is not asserted, fonts of different technologies will probably not match under any other criteria, either.

kFNSMatchGlyphs

Glyph repertoires and outline/bitmap data must match.

kFNSMatchEncodings

All cmaps in the font must match. Cmap reordering is also flagged as a mismatch because this can cause QuickDraw Text to use a different one.

kFNSMatchQDMetrics

Metrics used by QuickDraw Text must match. This includes the effect of fractEnable and any metric information in the FOND. The effect of hints at different sizes are taken into account by capturing the hint data itself.

kFNSMatchATSUMetrics

Metrics used by ATSUI must match. This includes both horizontal and vertical metrics. The effect of hints at different sizes are taken into account by capturing the hint data itself.

kFNSMatchKerning

Kerning data must match. This includes an sfnt's 'kern' table, the optional kerning data in a FOND, and any kerning data in a PostScript font's AFM file.

kFNSMatchWSLayout

Layout information given by an itl5, whether attached directly to the font or the one provided in the script bundle, must match.

kFNSMatchAATLayout

Advanced layout information, such as that used by ATSUI, must match. This includes such things as ligature and morph tables. In FontSync 1.0, we include OpenType-style layout information in this option.

kFNSMatchPrintEncoding

PostScript names and FOND re-encoding vectors must match. Note that it is an error for a font's internal PostScript name to be different from the one in the FOND, but FontSync will record both and consider them separately.

kFNSMissingDataNoMatch

If both references being compared do not have sufficient data to determine if they match, consider this a mismatch. This sense of the bit is chosen to make `kFNSMatchAll` specify the most stringent possible match criteria.

kFNSMatchAll

All attributes which FontSync considers must match. This also asserts `kFNSMissingDataNoMatch`, above.

At least one of the above options must be specified. Having all of these flags clear is tantamount to saying "don't look at anything," so any font will match. Because of this, zero has a special meaning described below under `kFNSMatchDefaults`. Setting undefined bits, on the other hand, will not generate an error. It does no immediate harm and allows some degree of backward compatibility.

kFNSMatchDefaults

Use the global default match criteria established by the API below (or set by the user via the control panel). Since having all flags clear is nonsensical as, 0 was chosen to represent this option.

5.1.3 Profile Type and Creator

```
enum {
      kFNSCreatorDefault =          0,
      kFNSProfileFileType =         'fnsp'
};
```

kFNSCreatorDefault

This is the file creator code to be used with `FNSProfileCreate` (§5.4.1) if you want to let FontSync assign a creator code instead of using one of your own. The actual creator code used in 1.0 will be ‘`fns `’ (the appe’s signature).

kFNSProfileFileType

This is the file type of a FontSync profile. It is not used directly in the API, but is provided for convenience, e.g., for putting up a Nav Services dialog to select a profile.

5.1.4 Versioning

```
enum unsigned long FNSObjectVersion {
      kFNSVersionDontCare = 0,

      kFNSCurSysInfoVersion = 1,
};
```

kFNSVersionDontCare

There will almost certainly be changes made in the future to either the profile file format or the nature of the “fingerprints” in a font reference. Future versions of FontSync must of course be compatible with older data, but it may also be necessary to produce old versions of the data for use with older systems. To this end, all object creation functions take an explicit object version parameter. If an application wants the most recent version supported by the library regardless of its version number, it should pass in `kFNSVersionDontCare`. In FontSync 1.0, both font references and profiles are version 1.

kFNSCurSysInfoVersion

FNSGetSysInfo() uses a parameter block (described below) to return various version information about the currently-running FontSync to an application. To allow for future expansion of the information without introducing new API, this parameter block has a version number as its first field. The structure used by FontSync 1.0 is version 1.

```
struct FNSSysInfo {
      FNSObjectVersion           iSysInfoVersion;
      FNSFeatureFlags            oFeatures;
      FNSObjectVersion           oCurRefVersion;
      FNSObjectVersion           oMinRefVersion;
      FNSObjectVersion           oCurProfileVersion;
      FNSObjectVersion           oMinProfileVersion;
      UInt16                     oFontSyncVersion;
};
```

FNSSysInfo

This is the parameter block used by `FNSSysInfoGet` (§5.7.1). The first field is an input to FontSync which tells it which version of the parameter block structure the caller is using (currently 1). The rest of the fields are outputs: `oFeatures` gives the feature flags (none are defined in FontSync 1.0); `oCurRefVersion` and `oMinRefVersion` give the current font reference structure version and the oldest one supported by the library, respectively; `oCurProfileVersion` and `oMinProfileVersion` give the same information for profiles;

`oFontSyncVersion` is a binary-coded decimal value. The high-order 8 bits give the major version, the next four give the minor version, and the last four give the revision. Version 1.0 would be encoded as `0x0100`. If this structure changes in the future, the changes will consist of additional fields added to the end of the structure. The version number (`kFNSCurSysInfoVersion`) will change accordingly.

5.1.5 Status Codes

```
enum {
      kFNSInvalidReferenceErr = -29580,
      kFNSBadReferenceVersionErr = -29581,
      kFNSInvalidProfileErr = -29582,
      kFNSBadProfileVersionErr = -29583,
      kFNSDuplicateReferenceErr = -29584,
      kFNSMismatchErr = -29585,
      kFNSInsufficientDataErr = -29586,
      kFNSBadFlattenedSizeErr = -29587,
      kFNSNameNotFoundErr = -29589,
};
```

kFNSInvalidReferenceErr

The font reference passed into the function was somehow invalid. This can also be returned by `FNSReferenceUnflatten` (§5.3.7) if the reconstructed reference is bad.

kFNSBadReferenceVersionErr

The font reference has an unsupported version number. This will generally happen if a reference created by a later version of FontSync is presented to an older one. It may also simply mean that the reference is invalid. A font reference's version number will always be found in the same place no matter what version it is. The validation code gives up as soon as it finds a bad version number. This error will also be returned from the font reference creation functions (§5.3.1 ff) if an unsupported reference version was requested.

kFNSInvalidProfileErr

The profile does not have a valid structure.

kFNSBadProfileVersionErr

The profile has an unsupported version number. Just as with a bad font reference version number, this may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid.

kFNSDuplicateReferenceErr

This status code is returned by `FNSProfileAddReference` (§5.4.7) when the reference being added already exists in the profile.

kFNSMismatchErr

The general-purpose status code used by FontSync to indicate that a reference didn't match.

kFNSInsufficientDataErr

This status code will be returned by a match operation if `kFNSMissingDataNoMatch` has been requested and both references being compared are missing the same data.

kFNSBadFlattenedSizeErr

Returned by `FNSReferenceUnflatten` (§5.3.7) if the passed-in size doesn't match the size recorded in the flattened reference or if the size was not large enough for a flattened reference.

kFNSNameNotFoundErr

There was no name in the font reference that matched the given parameters.

5.2 Default Match Criteria

Note that there is no API for setting the default match criteria.If desired, this may be accomplished by sending an appropriate Apple Event to the FontSync control panel (see §6).

5.2.1 FNSMatchDefaultsGet

```
FNSMatchOptions      FNSMatchDefaultsGet( void );
```

This function gets the bit mask used when `kFNSMatchDefaults` is passed to functions which perform matching. This value is read from a preferences file maintained by the FontSync library. If there is no preferences file, or it is unreadable, an implementation-defined fallback value will be returned. In FontSync 1.0, it has all defined options turned on.

Return Values

Match options bit mask.

5.3 Working with Font References

5.3.1 FNSReferenceCreate

```
OSStatus      FNSReferenceCreate( FMFont iFont,
                        FNSObjectVersion iDesiredVersion,
                        FNSFontReference *oReference );
```

This function creates in the current heap a font reference for the given font object. This is the function an ATSUI application would use to create font references. If the given font object is associated with some font family, it will also include the QuickDraw Text-specific information from that associated family. *NB: This definition assumes that a font object can only belong to one family. FontSync will use whatever family is returned by FMGetFontFamilyInstanceFromFont.* The desired reference format version is specified by the parameter `iDesiredVersion`. It must be a version supported by the library, i.e., in the range reported by `FNSSysInfoGet` (§5.7.1). If no particular version is required, the caller should pass `kFNSVersionDontCare`.

Return Values

`noErr`, `paramErr`, `kFMInvalidFontErr`, `kFNSBadReferenceVersionErr`, Memory Manager errors.

5.3.2 FNSReferenceCreateFromFamily

```
OSStatus      FNSReferenceCreateFromFamily( FNFontFamily iFamily,
                                  Style iStyle,
                                  FNSObjectVersion iDesiredVersion,
                                  FNSFontReference *oReference,
                                  Style *oActualStyle );
```

This function creates in the current heap a font reference based on the given font family and style. This is the function a classic QuickDraw Text application would use to create font references. Note that a particular QuickDraw style request is not necessarily fully satisfied by a “real” font. There are often left-over style bits. For example, a family may not have a real italic strike, so any italicization is handled by skewing the glyphs. The “real” style that the returned font reference corresponds to is returned in `oActualStyle`. The same information may be retrieved from the reference itself, but making it a function parameter helps to remind callers that the reference they’re getting may not be for the exact combination of style bits passed into this function. This parameter may be `NULL`. The desired reference format version is specified by the parameter `iDesiredVersion`. It must be a version supported by the library, i.e., in the range reported by `FNSSysInfoGet` (§5.7.1). If no particular version is required, the caller should pass `kFNSVersionDontCare`.

Return Values

`noErr`, `paramErr`, `kFNSBadReferenceVersionErr`, `kFMInvalidFontFamilyErr`, Memory Manager errors.

5.3.3 FNSReferenceDispose

```
OSStatus      FNSReferenceDispose( FNSFontReference iReference );
```

Disposes of the storage used by the given font reference. The reference should not be used any more after this call.

Return Values

`noErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, Memory Manager errors.

5.3.4 FNSReferenceMatch

```
OSStatus      FNSReferenceMatch( FNSFontReference iReference1,
                                 FNSFontReference iReference2,
                                 FNSMatchOptions iOptions,
                                 FNSMatchOptions *oFailedMatchOptions );
```

Compares the specified font references against each other using the specified options. If the match fails, `oFailedMatchOptions` which elements failed to match. This parameter may be `NULL` if details of the mismatch are not desired.

Return Values

`noErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSMismatchErr`, `kFNSInsufficientDataErr`, Memory Manager errors.

5.3.5 FNSReferenceFlattenedSize

```
OSStatus      FNSReferenceFlattenedSize( FNSFontReference iReference,
                                         ByteCount *oFlattenedSize );
```

Computes the space required for the flattened form of the font reference. This can also be computed by passing a `NULL` `iFlatReference` pointer to `FNSReferenceFlatten`, below. We include an explicit call to emphasize that both steps are usually required when flattening a font reference.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`,
`kFNSBadReferenceVersionErr`

5.3.6 FNSReferenceFlatten

```
OSStatus      FNSReferenceFlatten( FNSFontReference iReference,
                         void *iFlatReference,
                         ByteCount *oFlattenedSize );
```

Flattens the specified font reference into a form which can be stored externally. It returns the size of the flattened reference in `oFlattenedSize`. Unlike many similar APIs, this function will not do "partial writes." It assumes that the storage pointed to by `iFlatReference` is large enough to hold the data and will always out a full flattened reference. This function can be called with `iFlatReference` set to `NULL` to just get the size of the flattened reference without creating one. Alteratively, `FNSReferenceFlattenedSize` can also be called to get this information.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`,
`kFNSBadReferenceVersionErr`

5.3.7 FNSReferenceUnflatten

```
OSStatus      FNSReferenceUnflatten( void *iFlatReference,
                             ByteCount iFlattenedSize,
                             FNSFontReference *oReference );
```

Reconstitutes a font reference from its external form. The size parameter is redundant since a flattened reference contains its own size, but allows a useful check to be made.

Return Values

`noErr`, `paramErr`, `kFNSBadFlattenedSizeErr`, `kFNSInvalidReferenceErr`,
`kFNSBadReferenceVersionErr`, Memory Manager errors.

5.3.8 FNSReferenceGetVersion

```
OSStatus      FNSReferenceGetVersion( FNSFontReference iReference,
                             FNSObjectVersion *oVersion );
```

Returns the format version number of the given font reference.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`.

5.4 Working With Font Profiles

Important: FontSync profiles are always housed in a file. FontSync attempts to moderate access to this file. Ideally, it tries to either allow many readers or exactly one writer but not both. Unfortunately, the Mac OS File Manager does not allow this kind of exclusion on local volumes, so it may still be possible for someone to get write access to a profile when there are active readers. Rather than complicating the implementation to work around this limitation, we choose to treat FontSync profile files as most any other document file. That is, we make it the caller's problem. If you are going to modify a profile, the safest way to do it is to first make a copy of the file, make the modifications to the copy, and swap file names when done. This method has the added benefit of preserving the original profile in the case of a catastrophic error that leaves the new profile invalid.

5.4.1 FNSProfileCreate

```
OSStatus      FNSProfileCreate( const FSSpec *iFile,
                                FourCharCode iCreator,
                                ItemCount iEstNumRefs,
                                FNSObjectVersion iDesiredVersion,
                                FNSFontProfile *oProfile );
```

This function creates and opens (with read/write access) the given file and initializes it to an empty FontSync profile. The file's creator will be iCreator. If you don't want to specify a creator code of your own, pass in `0` (`kFNSCreatorDefault`). The new profile will usually immediately have font references added to it, so to minimize the number of times the file needs to be grown, the estimated number of references that it will contain can be provided in `iEstNumRefs`. This parameter can be zero. The desired profile format version is specified by the parameter `iDesiredVersion`. It must be a version supported by the library, i.e., in the range reported by `FNSSysInfoGet` (§5.7.1). If no particular version is required, the caller should pass `kFNSVersionDontCare`. The returned `FNSFontProfile` is a "cookie" that identifies that profile to FontSync.

Return Values

> `noErr`, `paramErr`, `kFNSBadProfileVersionErr`, File Manager errors, Memory Manager errors.

5.4.2 FNSProfileOpen

```
OSStatus      FNSProfileOpen( const FSSpec *iFile,
                              Boolean iOpenForWrite,
                              FNSFontProfile *oProfile );
```

This function returns a profile "cookie" for an existing FontSync profile file. If the profile is going to be edited it should be opened for read/write access by setting `iOpenForWrite` to `true`.

Return Values

> `noErr`, `paramErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, File Manager errors, Memory Manager errors.

5.4.3 FNSProfileClose

```
OSStatus      FNSProfileClose( FNSFontProfile oProfile );
```

Closes the file associated with the given profile "cookie" and marks the cookie as invalid. Any memory associated with the open profile is released. Note that closing a profile does not automatically compact it. An explicit call to `FNSProfileCompact` must be made (before closing the profile, of course) to do so.

Return Values

> `noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, File Manager errors.

5.4.4 FNSProfileCompact

```
OSStatus      FNSProfileCompact( FNSFontProfile oProfile );
```

While a profile is being constructed it will contain “excess capacity” to reduce the number of times that the file must be grown and data moved around. This function will eliminate that excess space and minimize the size of the file. This will typically be called just before closing a profile that has been edited. If called on a profile that has not been opened for write, the function will simply return without doing anything.

Return Values

> `noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, File Manager errors, Memory Manager errors.

5.4.5 FNSProfileClear

```
OSStatus      FNSProfileClear( FNSFontProfile oProfile );
```

Removes all references from the profile. The profile must be writable. The file remains the same size after this operation.

Return Values

> `noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, `permErr`, File Manager errors, Memory Manager errors.

5.4.5 FNSProfileCountReferences

```
OSStatus      FNSProfileCountReferences( FNSFontProfile iProfile,
                            ItemCount *oFontCount );
```

Returns the number of references in the profile.

Return Values

> `noErr`, `paramErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`.

5.4.6 FNSProfileGetIndReference

```
OSStatus      FNSProfileGetIndReference( FNSFontProfile iProfile,
                            UInt32 iWhichReference,
                            FNSFontReference *oReference );
```

Returns a font reference indexed by `iWhichReference`; the range is from 0 to one less than the number of references (given by `FNSProfileCountReferences`) in the profile.

Return Values

> `noErr`, `paramErr`, `inputOutOfBounds`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, File Manager errors, Memory Manager errors.

5.4.7 FNSProfileAddReference

```
OSStatus      FNSProfileAddReference( FNSFontProfile iProfile,
                                  FNSFontReference iReference );
```

Adds the specified font reference to the profile. The profile must be writable. If an identical (not just matching) reference already exists in the profile, a new one will not be added and `kFNSDuplicateReferenceErr` will be returned. *NB: In FontSync 1.0, we expect that new* *profiles will simply overwrite old ones wholesale. If the capability for incremental updates is required (e.g. to add data to a particular reference) it will be provided in a future version.*

Return Values

`noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSDuplicateReferenceErr`, `permErr`, other File Manager errors, Memory Manager errors.

5.4.8 FNSProfileRemoveReference

```
OSStatus      FNSProfileRemoveReference( FNSFontProfile iProfile,
                                         FNSFontReference iReference );
```

Deletes the specified font reference from the profile. This will delete only an identical reference, not one that simply matches the given reference. The profile must be writable.

Return Values

`noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSMismatchErr`, `permErr`, other File Manager errors, Memory Manager errors.

5.4.9 FNSProfileRemoveIndReference

```
OSStatus      FNSProfileRemoveIndReference( FNSFontProfile iProfile,
                                            UInt32 iIndex );
```

Deletes the `iIndex`th font reference from the profile; the range is from 0 to one less than the number of references (given by `FNSProfileCount`) in the profile. Note that this will change the indices of all succeeding references. The profile must be writable.

Return Values

`noErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `inputOutOfBounds`, `permErr`, other File Manager errors, Memory Manager errors.

5.4.10 FNSProfileGetVersion

```
OSStatus      FNSProfileGetVersion( FNSFontProfile iProfile,
                                    FNSObjectVersion *oVersion );
```

Returns the format version number of the given open font profile.

Return Values

`noErr`, `paramErr`, `kFNSInvalidProfileErr`.

5.5 Searching by Font Reference

5.5.1 FNSProfileMatchReference

```
OSStatus      FNSProfileMatchReference( FNSFontProfile iProfile,
                                        FNSFontReference iReference,
                                        FNSMatchOptions iMatchOptions,
                                        ItemCount iOutputSize,
                                        UInt32 oIndices[],
                                        ItemCount *oNumMatches );
```

This function returns a list of font references in the given profile which match the specified reference. The list will contain indices that can be used with `FNSProfileIterate` to retrieve the actual matching font references. To simply count the number of matches, call this function with `iOutputSize` set to zero. This function will fill in as many elements of `oIndices` as it can, but never more than `iOutputSize`. The total number of matches will always be returned in `oNumMatches` regardless of how many entries it made in `oIndices`. To simply determine if there is a match at all, call this function with both `iOutputSize` set to zero and `oNumMatches` `NULL` and test for a status of `noErr` (matches found) or `kFNSMismatchErr` (no matches). *We use a bulk operator like this instead of some other style of iterator because we expect the number of matches to be small.*

Return Values

> `noErr`, `paramErr`, `kFNSInvalidProfileErr`, `kFNSBadProfileVersionErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSMismatchErr`, File Manager errors, Memory Manager errors.

5.5.2 FNSReferenceMatchFonts

```
OSStatus      FNSReferenceMatchFonts( FNSFontReference iReference,
                                    FNSMatchOptions iMatchOptions,
                                    ItemCount iOutputSize,
                                    FMFont oFonts[],
                                    ItemCount *oNumMatches );
```

This function returns a list of active fonts which match the specified reference. It is the function to be used to map a font reference to an actual font. There may be more than one such font, so a list is returned. To simply count the number of matches, call this function with `iOutputSize` set to zero. This function will fill in as many elements of `oFonts` as it can, but never more than `iOuputSize`. The total number of matches will always be returned in `oNumMatches` regardless of how many entries it made in `oIndices`. If FontSync finds no matching fonts it may query a third party to find matches. See §8 for details. *We use a bulk operator like this instead of some other style of iterator because we expect the number of matches to be small.*

Return Values

> `noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSMismatchErr`, Memory Manager errors.

5.5.3 FNSReferenceMatchFamilies

```
OSStatus      FNSReferenceMatchFamilies( FNSFontReference iReference,
                                    FNSMatchOptions iMatchOptions,
                                    ItemCount iOutputSize,
                                    FMFontSpecification oFonts[],
                                    ItemCount *oNumMatches );
```

This function is identical to the previous one, except that it returns a list of font families instead of font objects. This is the function to use to convert a font reference into an actual font that can be used with QuickDraw Text.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSMismatchErr`, Memory Manager errors.

5.6 UI Support

5.6.1 FNSReferenceGetFamilyInfo

```
OSStatus      FNSReferenceGetFamilyInfo( FNSFontReference iReference,
                                Str255 oFamilyName,
                                ScriptCode *oFamilyNameScript,
                                Style *oActualStyle );
```

Returns information about the font family associated with the reference: `oFamilyName` is the name by which the font is known to the classic Font Manager (i.e., the string to pass to `GetFNum`). `oFamilyNameScript` is the script of the family name string. `oActualStyle` gives the QuickDraw style associated with the font reference. This is the style actually represented by the reference, not necessarily the one passed into `FNSReferenceCreateFromFamily` (see §5.3.2). Any of the output pointers may be `NULL`, but a `paramErr` will result if they all are.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSInsufficientDataErr`.

5.6.2 FNSReferenceCountNames

```
OSStatus      FNSReferenceCountNames( FNSFontReference iReference,
                                ItemCount *oNameCount );
```

`oNameCount` gives a count of the (internal) font names, other than the `GetFNum` family name, recorded in the given reference. These names will always include the PostScript and unique names, if available.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSInsufficientDataErr`.

5.6.3 FNSReferenceGetIndName

```
OSStatus      FNSReferenceGetIndName( FNSFontReference iReference,
                                ItemCount iFontNameIndex,
                                ByteCount iMaximumNameLength,
                                Ptr oName,
                                ByteCount *oActualNameLength,
                                FontNameCode *oFontNameCode,
                                FontPlatformCode *oFontNamePlatform,
                                FontScriptCode *oFontNameScript,
                                FontLanguageCode *oFontNameLanguage );
```

Retrieves the `iFontNameIndexth` name recorded in the given font reference. The conventions for and interpretations of the parameters are the same as for the ATSUI function `ATSUGetIndFontName`.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSInsufficientDataErr`, `inputOutOfBounds`.

5.6.4 FNSReferenceFindName

```
OSStatus      FNSReferenceFindName( FNSFontReference iReference,
                                    FontNameCode iFontNameCode,
                                    FontPlatformCode iFontNamePlatform,
                                    FontScriptCode iFontNameScript,
                                    FontLanguageCode iFontNameLanguage,
                                    ByteCount iMaximumNameLength,
                                    Ptr oName,
                                    ByteCount *oActualNameLength,
                                    ItemCount *oFontNameIndex );
```

Finds the first name that matches the given specification, if any. The conventions for and interpretations of the parameters are the same as for the ATSUI function `ATSUGetIndFontName`. Any of the name code, platform, script or language parameters may be their wildcard values, though using a wildcard for the name code rarely makes sense.

Return Values

`noErr`, `paramErr`, `kFNSInvalidReferenceErr`, `kFNSBadReferenceVersionErr`, `kFNSInsufficientDataErr`, `kFNSNameNotFoundErr`.

5.7 Miscellany

5.7.1 FNSSysInfoGet

```
void   FNSSysInfoGet( FNSSysInfo *ioInfo );
```

Returns version and feature information for the version of FontSync running in the current context. See §5.1.4 for a description of the parameter block.

5.7.2 FNSEnabled

```
Boolean       FNSEnabled( void );
```

Returns a flag that indicates whether or not FontSync operations should be performed by an application. This flag has no effect on the operation of any FontSync API, but should be checked by applications before starting a sequence of FontSync calls. Note that there is no way in FontSync 1.0 to set this flag to anything but true. This function is left over from previous iterations of the HI design. The ability to disable FontSync without removing the extension will probably reappear in a future version, so we left this function in place so that applications won’t have to change when the functionality is enabled.

Return Values

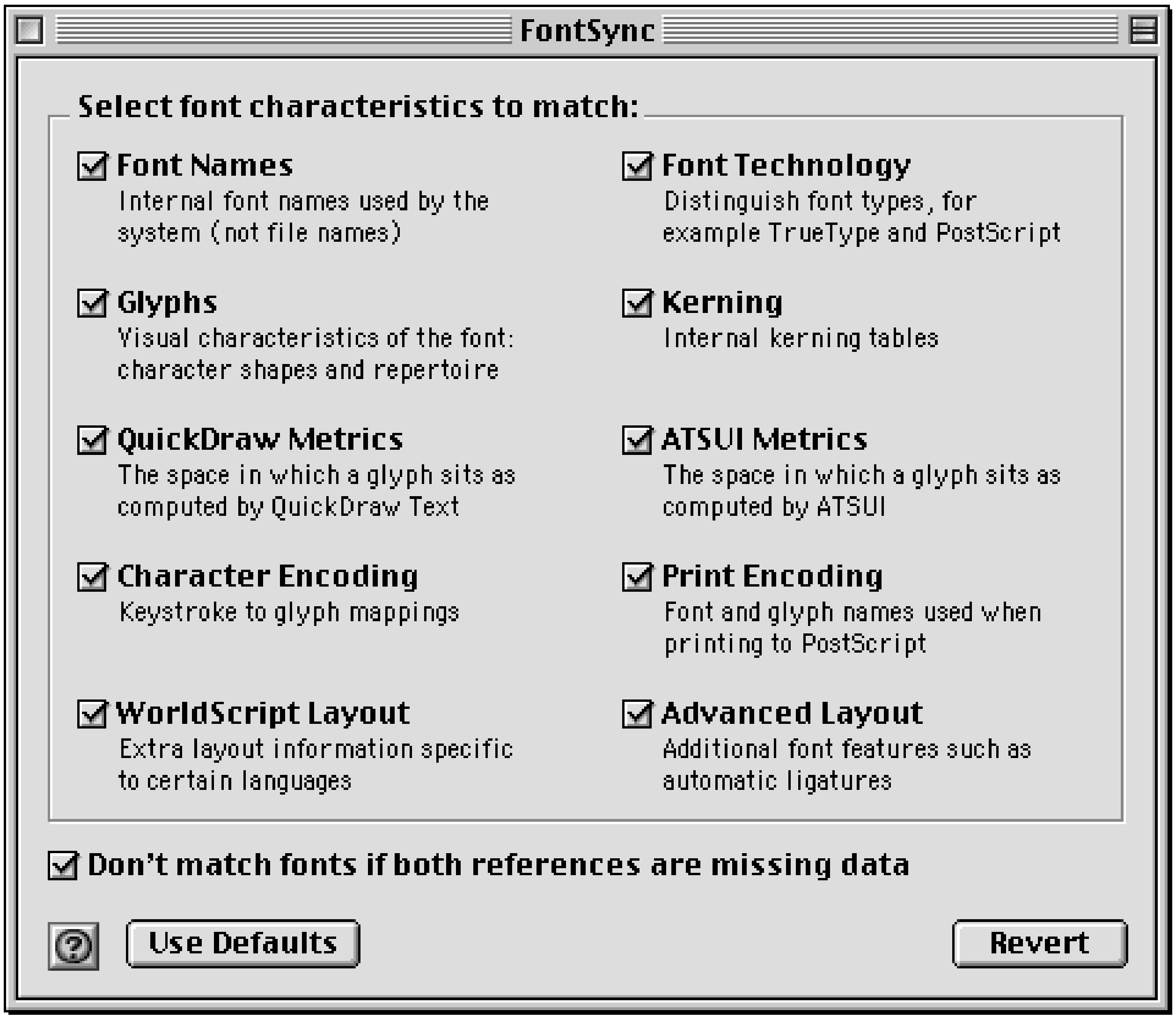
A flag indicating whether or not FontSync operations should be performed by an application.

6.0 Control Panel

There will not be a detailed HI spec for the FontSync control panel. This section of the ERS describes what it should look like and what functionality it should have.

For most people, the out-of-the-box settings should be sufficient. For advanced users who either need or want to change those settings, a optionally-installed control panel provides two ways of doing so. A user can either change the setting directly in the control panel or run a script that does so. This control panel also serves as a model UI for applications that want to give users this same level of control directly in the application.

The control panel consists of a single dialog (illustrated below). For each defined match option, there is a checkbox in the dialog. There are two classes of options: the bulk of them represent parts of a font that must match. These will be grouped together visually by putting a box around them. There are enough of these that they will probably need to be arranged in two columns to keep the dialog from getting too tall. Since it doesn't make sense to turn all of these off (any font would match in that case) attempting to uncheck the last enabled option should post an alert explaning why that is prohibited. The remaining checkbox in this dialog corresponds to the `kFNSMissingDataNoMatch` flag, which is independent of the other match criteria. Its checkbox will be outside the group box described above. All these checkboxes will have a short label as well as a couple of descriptive sentences under that in a smaller font. These checkboxes are "live," that is, the changes take effect immediately. In addition to the checkboxes, this dialog should have buttons to cancel the changes and to set them to the implementation-defined defaults. If the prefs file could not be written, a suitable alert should be displayed.



FontSync

Select font characteristics to match:

- ☑ **Font Names**
  Internal font names used by the system (not file names)
- ☑ **Font Technology**
  Distinguish font types, for example TrueType and PostScript
- ☑ **Glyphs**
  Visual characteristics of the font: character shapes and repertoire
- ☑ **Kerning**
  Internal kerning tables
- ☑ **QuickDraw Metrics**
  The space in which a glyph sits as computed by QuickDraw Text
- ☑ **ATSUI Metrics**
  The space in which a glyph sits as computed by ATSUI
- ☑ **Character Encoding**
  Keystroke to glyph mappings
- ☑ **Print Encoding**
  Font and glyph names used when printing to PostScript
- ☑ **WorldScript Layout**
  Extra layout information specific to certain languages
- ☑ **Advanced Layout**
  Additional font features such as automatic ligatures

☑ **Don't match fonts if both references are missing data**

The figure above is illustrative and should not be taken as an exact layout.

The control panel should be scriptable (see §7 for details). If the FontSync library is not available, the control panel should fail gracefully and post an appropriate alert.

7.0 Scripting

We expect FontSync to be an important component of publishing workflow. AppleScript is used very heavily in that environment, both directly by users and as components of cgi gateways. It is thus imperative that FontSync be somehow accessible from scripts.

The control panel (§6) will allow setting the default match criteria both directly and through a script. Each item that can be set is a property of the FontSync CP application. Settings can thus be made via the standard Set Data event. Current settings can be fetched via Get Data. Since there is no API for setting default match criteria, applications that wish to do this will also have to send Apple Events to the control panel. Note that the optional control panel must be installed to have access to this feature. Applications can always get the current settings directly via an API. For convenience, there will be a “bulk” mode for these events.

In addition, FontSync 1.0 will support Apple Events that create and compare profiles. This support will be provided by a faceless background ‘appe’ application. Two new Apple Events defined for this: Create Profile and Compare Profile. Create Profile takes an alias to a file and creates a profile from the currently-active fonts. The reply simply indicates success or failure. Compare Profile takes an alias to an existing profile and compares the currently-active fonts to it. The reply is a list of “problems.” Compare Profile can take an optional second argument which is also a profile. If this argument is present, the second profile will be compared instead of the active fonts.

There is no way to construct a FontSync font reference via Apple Events in version 1.0. Doing so requires solving the context ambiguity problem. There does not appear to be a need to provide such a facility yet, though as users gain experience with FontSync we anticipate having to add such capabilities.

There is a difficult issue around FontSync scriptability: ambiguity of context. Font family IDs (and perhaps font object IDs) can be local to a process. Outside of that process, the ID may either be invalid or refer to another font. For example, the ID assigned to a font embedded in an application is only valid in that application context. In MacOS X, there may be another layer besides this (fonts visible only to a particular user). The Apple Events described above will generally be sent to another process. There is currently no good way to have the Apple Events always processed in the same context as the sender. That is, there are ways to do so, but they reduce the robustness of the system. That means that the profile creation and comparison done above will be done based on the fonts visible from the faceless background application responding to the events. This is a relatively rare occurence, so for the simple functionality envisioned for version 1.0, using a different context is sufficient.

7.1 FontSync Settings Suite

The FontSync Control Panel supports a subset of the required and core suites. Specifically, the following events are supported:

Open Application
Quit Application
Get Data
Set Data

These events are identical to the standard ones in the Apple Event Registry. Two object classes are defined in this suite: Application and Match Options.

7.1.1 Application class

This class is a subset of the standard application class defined in the Apple Event Registry. It has only the following properties:

name

Tag:	`'fnsp'`
Class:	International Text
Description:	the application’s name
Modifiable:	no

frontmost

Tag:	`'pisf'`
Class:	Boolean
Description:	Is this the frontmost application?
Modifiable:	no

version

Tag:	`'vers'`
Class:	Version
Description:	the version of the application
Modifiable:	no

default match options

Tag:	`'fnos'`
Class:	Match Options
Description:	the default criteria to use when matching font references
Modifiable:	yes

properties

Tag:	`'pAll'`
Class:	Apple Event Record
Description:	all of the application’s properties in a record
Modifiable:	only elements which correspond to modifiable properties.

7.1.2 Match Options class

This class represents a value of type `FNSMatchOptions`. Its structure is similar to a Text Styles object. That is, it is a record consisting of two lists, one of which represents the options that are on, the other the options that are off. These lists may be incomplete, in which case the object is interpreted as representing a delta from some fixed value. It is an error for the same option to appear in both the on and off lists. Match Options may also be represented as long integers, which of course fully specify the on and off options. Standard coersions are provided to convert between these two representations.

Properties:

on options

Tag:	`'onmo'`
Class:	List of `'fnsm'` enum.
Description:	the matching options that are on
Modifiable:	yes

off options

Tag:	`'ofmo'`
Class:	List of `'fnsm'` enum.
Description:	the matching options that are off
Modifiable:	yes

The `'fnsm'` enumeration has a member for each defined bit in an `FNSMatchOptions` value. The value for each enumerator is a pseudo-four character code which consists of the three prefix characters `'fno'` followed by $\log_2$ of the bit value. For example, the value corresponding to `kFNSMatchNames` is `0x666e6f00`, while the value corresponding to `kFNSMatchEncodings` is `0x666e6f03`.

The following table lists the enum names for each of the `FNSMatchOptions` bits:

`kFNSMatchNames`	font names
`kFNSMatchTechnology`	font types
`kFNSMatchGlyphs`	glyphs
`kFNSMatchEncodings`	encodings
`kFNSMatchQDMetrics`	QuickDraw metrics
`kFNSMatchATSUMetrics`	ATSUI metrics
`kFNSMatchKerning`	kerning
`kFNSMatchWSLayout`	WorldScript layout
`kFNSMatchAATLayout`	advanced layout
`kFNSMatchPrintEncoding`	print encoding
`kFNSMissingDataNoMatch`	missing data mismatches

7.2 FontSync Suite

The FontSync suite is implemented by a faceless background application packaged with the shared library. It provides a small subset of the required and core suites as well as adding events for creating and matching FontSync profiles. In the future, this suite will be expanded to allow direct manipulation of font references and profiles.

The Open Application, Quit Application, Get Data and Set Data events are just as described in the Apple Event Registry. In addition, the FontSync suite defines the following two events:

7.2.1 Create Font Profile

Event Class `'fns '`

Event ID `'mkfp'`

Parameters

---- (`keyDirectObject`)

Description:	the file in which to save the profile
Descriptor Type:	Alias
Required?	yes

with creator (`'fltp'`)

Description:	the creator code of the file in which to save the profile
Descriptor Type:	Type
Required?	no

version (`'vers'`)

Description:	the desired profile format version number
Descriptor Type:	Long Integer
Required?	no

Reply Parameters

None

This event will cause a profile of the currently-active fonts to be created. (From the point of view of the FBA — see the discussion in the introduction to this section.) If the creator code or version is not specified, the behavior of this event will follow the default behavior of FNSProfileCreate (§5.4.1).

7.2.2 Match Against

Event Class `'fns '`

Event ID `'mtfp'`

Parameters

---- (`keyDirectObject`)

Description:	the profile file to match against
Descriptor Type:	Alias
Required?	yes

using fonts from (`'fnff'`)

Description:	use the fonts in this profile instead of the current active fonts
Descriptor Type:	Alias
Required?	no

with match options (`'fnco'`)

Description:	matching options which override system default settings
Descriptor Type:	Match Options
Required?	no

Reply Parameters

A list of Match Result records or an error.

This event will compare the currently active fonts (in the FBA's context — see introduction) to the given profile. If a second profile is provided in an optional parameter, it will be used as the source of the fonts instead of the active fonts. The default match options may be overridden by specifying them in the “with match options” parameter. As described in §7.1.2, a value of type Match Options will be interpreted as a delta from the default options, while a value of type Long Integer will be taken as an absolute specification. The reply is a list of Match Result records, described in §7.2.4, below.

7.1.1 Application class

This class is a subset of the standard application class defined in the Apple Event Registry. It has the following properties:

name

Tag:	`'fnsp'`
Class:	International Text
Description:	the application’s name
Modifiable:	no

version

Tag:	`'vers'`
Class:	Version
Description:	the version of the application
Modifiable:	no

quit delay

Tag:	`'qdel'`
Class:	Element of the `'pDly'` enumeration
Description:	specifies how the application quits automatically
Modifiable:	yes

quit delay

Tag:	`'qdel'`
Class:	Long Integer
Description:	the time in seconds the application will idle before quitting
Modifiable:	yes

The name and version properties are standard. The two versions of the quit delay property are new (the idea is taken from ColorSync). FontSync attempts to be a good citizen by not leaving its faceless background application running longer than necessary. If it is left idle for some implementation-specified length of time, it will quit automatically.

This timeout can be controlled via the “quit delay” property. Positive values are interpreted as the number of seconds to idle before quitting. Non-positive numbers are members of the `'pDly'` enumeration. There are three such values defined:

name	**value**	**meaning**
immediate	0	quit immediately
default	-1	reset to the default timeout
never	-2	never time out

7.2.4 Match Result class

This class represents a mismatch or error when matching a particular font against a profile. It contains enough information to identify the font to the user. The class is a record with ID `'fnmr'`. It has the following elements:

problem reported

Tag:	`'stat'`
Class:	Member of the `'fnmp'` enumeration (see below)
Description:	The kind of problem reported
Modifiable:	no

name

Tag:	`'pnam'`
Class:	International text
Description:	The name of the problem font
Modifiable:	no

ID

Tag:	`'font'`
Class:	Short
Description:	The font family ID of the problem font. -1 if the font has no family.
Modifiable:	no

style

Tag:	`'txst'`
Class:	Short
Description:	The style of the problem font. Meaningless if ID is -1.
Modifiable:	no

If the font family ID is -1, the style element will not be present in the record.

The `'fnmp'` enumeration, which indicates the nature of the problem, consists of the following members:

mismatch	`'fnnm'`	the font didn't match
noRef	`'fnnr'`	couldn't create a font reference

8.0 Font Utility Hooks

As noted in §3, allowing third-party application to participate in FontSync's matching process is powerful. A simple and flexible way to accomplish this on Mac OS 8 is via an Apple Event-based protocol. This may not be appropriate for Mac OS X, however, so this aspect of FontSync may change in that environment.

The basic idea is this: When FontSync receives a request to find active fonts that match a font reference, it will try to satisfy that request on its own. If it cannot and someone has registered interest in this process, FontSync will send an Apple Event with the details of the request to this party. The receiver should respond with a list of matching fonts. It should take whatever steps are necessary to identify and activate them before replying to the event. Registration is handled by the simple expedient of installing a handler for the appropriate Apple Event. This handler will typically be installed in the system table, though FontSync will check for handlers both in the system and in the context's local handler table. The Apple Event will always be a send-to-self, which is reasonably efficient.

The details of the hook event are as follows:

Event Class `'fns '`

Event ID `'find'`

Parameters

`keyDirectObject`

Description:	the font reference to match
Descriptor Type:	flattened FontSync font reference
Required?	yes

`'fnco'`

Description:	the options to be used when matching
Descriptor Type:	Long Integer
Required?	yes

`'fnty'`

Description:	the desired reply type
Descriptor Type:	Integer
Required?	yes

Reply Parameters

List of either `FMFont` or `FMFontFamilyInstance` depending on the reply type parameter.

No user terminology is defined for this event, since it is completely visible to a user and is not intended for use in scripts.

The match options passed in this event are absolute, that is, meta-values have been resolved to the actual match option bits that should be in force.

The reply type parameter specifies whether the request was for font object IDs (0) or font family IDs (1). Other values may be defined in future versions of this hook.

The reply should be a list of matching fonts represented as either `FMFont` or `FMFontFamilyInstance` values, depending on the value of the desired type parameter. There are no semantics associated with the order of this list, though in many cases the application will just take the first element, so it is a good idea to put the “best” match first.

Appendix A: References

Applebaum, *FontSync 1.0 MRD*, 8 Feb 1999. Marketing requirements document for FontSync.

Developer Technical Publications, *Apple Event Registry: Standard Suites,* Winter 1992. Standard Apple Event descriptions.

Goldsmith and Opstad, *FontSync ERS Draft 7*, 20 Feb 1998. The original FontSync specification.

Mikawa, *Font Management ES*, version 0.3, 5 Feb 1999. Defines font objects and font families.

Appendix B: Change History

Draft 1 (5 Feb 1999): Based on *FontSync ERS Draft 7* by Goldsmith and Opstad, 20 Feb 1998.Updated API. Added (place holder) sections for scripting and control panel. Eliminated font iterators and match options from profiles. Made profiles file-based. Added requirements and control panel description.

Draft 2 (15 Feb 1999): Changed a few function names to match the style of the rest of the API. Added status codes. Fixed some typos. Added creator code to `FNSProfileCreate`. Changes per review: clarified some things; replaced Gestalt selectors with explicit API; added explicit flattened size API; added permissions to `FNSProfileOpen`; filled out list of match criteria. Removed enable/disable option from CP and supporting API stuff — if people want to disable FontSync they can just use the Extension Manager.

Draft 3 (25 Feb 1999): Added missing `FNSClearProfile`() description. Moved “Set Match Options” from the CP menu to the main dialog. Noted which profile operations require the profile to be writable.

Draft 4 (10 Mar 1999): Changed default creator code to ‘fnsc’ to make the CP’s bundle easier to deal with. Allow multiple profiles to be selected for the CP’s “Compare Profiles” function. Added sample screen shots of the main and “Set Match Criteria” dialogs. Clarified handling of invalid match options. Added `oActualStyle` return value to `FNSReferenceCreateFromProfile`.

Draft 5 (9 Apr 1999): Added status codes. Removed `FNSMatchDefaultsSet`. Allow calling `FNSProfileMatchReference` with no outputs for a simple yes/no query. Finally documented the split of `FNSReferenceGetNameInfo` into `FNSReferenceCountNames` and `FNSReferenceGetFamilyInfo`. Changed prototype of `FNSVersionInfo`. Made the version code a 16-bit BCD value. Added `FNSEnabled`. Added a warning regarding the lack of access protection to profiles.

Draft 6 (19 Apr 1999): One more round of repackaging. The CP is now off by itself. Added a new mockup of the options dialog: the text and layout are pretty close to final. Sketched out the scripting support that will be provided. Detailed specs for the Apple Events will be in the next draft. Added “Deliverables” section.The shlb gets packaged with an ‘appe’ that implements the scripting support.

Draft 7 (30 Aug 1999): Filled in details of the Apple Event suite. Described third-party matcher hook. Interface changes: kFNSMatchATSULayout became kFNSMatchAATLayout; explicit versioning added; removed kFNSInvalidFontErr – use kFMInvalidFontErr instead; FNSVersionInfo became FNSSysInfoGet. Default profile creator code changed. CP checkboxes are live. Minor tweak to CP layout.

Appendix C: Sample Code

Later.