Core Text Reference Collection

Data Management: Strings, Text, & Fonts

2010-02-25

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Core Text Reference Collection

Framework	/System/Library/Frameworks/ApplicationServices.framework/
Header file directories	/System/Library/Frameworks/ApplicationServices.framework/Headers
Companion guide	Core Text Programming Guide
Declared in	CTFont.h CTFontCollection.h CTFontDescriptor.h CTFontTraits.h CTFrame.h CTFramesetter.h CTGlyphInfo.h CTLine.h CTParagraphStyle.h CTRun.h CTRunDelegate.h CTStringAttributes.h CTTextTab.h CTTypesetter.h CoreText.h

This collection of documents is the API reference for the Core Text framework. Core Text provides a modern, low-level programming interface for laying out text and handling fonts. The Core Text layout engine is designed for high performance, ease of use, and close integration with Core Foundation. The text layout API provides high-quality typesetting, including character-to-glyph conversion, with ligatures, kerning, and so on. The complementary Core Text font technology provides automatic font substitution (cascading), font descriptors and collections, easy access to font metrics and glyph data, and many other features.

Multicore Considerations: All individual functions in Core Text are thread safe. Font objects (CTFont, CTFontDescriptor, and associated objects) can be used by simultaneously by multiple operations, work queues, or threads. However, the layout objects (CTTypesetter, CTFramesetter, CTRun, CTLine, CTFrame, and associated objects) should be used in a single operation, work queue, or thread.

Core Text Reference Collection

Opaque Types

PART I

Opaque Types

CTFont Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTFont.h

Overview

The CTFont opaque type represents a Core Text font object. Font objects represent fonts to an application, providing access to characteristics of the font, such as point size, transform matrix, and other attributes. Fonts provide assistance in laying out glyphs relative to one another and are used to establish the current font when drawing in a graphics context.

Functions by Task

Creating Fonts

`CTFontCreateWithName` (page 28)
Returns a new font reference for the given name.

`CTFontCreateWithNameAndOptions` (page 29)
Returns a new font reference for the given name.

`CTFontCreateWithFontDescriptor` (page 27)
Returns a new font reference that best matches the given font descriptor.

`CTFontCreateWithFontDescriptorAndOptions` (page 27)
Returns a new font reference that best matches the given font descriptor.

`CTFontCreateUIFontForLanguage` (page 26)
Returns the special user-interface font for the given language and user-interface type.

`CTFontCreateCopyWithAttributes` (page 23)
Returns a new font with additional attributes based on the original font.

`CTFontCreateCopyWithSymbolicTraits` (page 24)
Returns a new font in the same font family as the original with the specified symbolic traits.

`CTFontCreateCopyWithFamily` (page 24)
Returns a new font in the specified family based on the traits of the original font.

`CTFontCreateForString` (page 25)
Returns a new font reference that can best map the given string range based on the current font.

Getting Font Data

`CTFontCopyFontDescriptor` (page 18)
Returns the normalized font descriptor for the given font reference.

`CTFontCopyAttribute` (page 14)
Returns the value associated with an arbitrary attribute of the given font.

`CTFontGetSize` (page 35)
Returns the point size of the given font.

`CTFontGetMatrix` (page 35)
Returns the transformation matrix of the given font.

`CTFontGetSymbolicTraits` (page 36)
Returns the symbolic traits of the given font.

`CTFontCopyTraits` (page 21)
Returns the traits dictionary of the given font.

Getting Font Names

`CTFontCopyPostScriptName` (page 20)
Returns the PostScript name of the given font.

`CTFontCopyFamilyName` (page 16)
Returns the family name of the given font.

`CTFontCopyFullName` (page 18)
Returns the full name of the given font.

`CTFontCopyDisplayName` (page 16)
Returns the display name of the given font.

`CTFontCopyName` (page 20)
Returns a reference to the requested name of the given font.

`CTFontCopyLocalizedName` (page 19)
Returns a reference to a localized name for the given font.

Working With Encoding

`CTFontCopyCharacterSet` (page 15)
Returns the Unicode character set of the font.

`CTFontGetStringEncoding` (page 36)
Returns the best string encoding for legacy format support.

`CTFontCopySupportedLanguages` (page 20)
Returns an array of languages supported by the font.

`CTFontGetGlyphsForCharacters` (page 33)
Provides basic Unicode encoding for the given font, returning by reference an array of `CGGlyph` values corresponding to a given array of Unicode characters for the given font.

Getting Font Metrics

`CTFontGetAscent` (page 30)
Returns the scaled font-ascent metric of the given font.

`CTFontGetDescent` (page 32)
Returns the scaled font-descent metric of the given font.

`CTFontGetLeading` (page 34)
Returns the scaled font-leading metric of the given font.

`CTFontGetUnitsPerEm` (page 38)
Returns the units-per-em metric of the given font.

`CTFontGetGlyphCount` (page 33)
Returns the number of glyphs of the given font.

`CTFontGetBoundingBox` (page 31)
Returns the scaled bounding box of the given font.

`CTFontGetUnderlinePosition` (page 37)
Returns the scaled underline position of the given font.

`CTFontGetUnderlineThickness` (page 38)
Returns the scaled underline-thickness metric of the given font.

`CTFontGetSlantAngle` (page 36)
Returns the slant angle of the given font.

`CTFontGetCapHeight` (page 32)
Returns the cap-height metric of the given font.

`CTFontGetXHeight` (page 39)
Returns the x-height metric of the given font.

Getting Glyph Data

`CTFontCreatePathForGlyph` (page 25)
Creates a path for the specified glyph.

`CTFontGetGlyphWithName` (page 34)
Returns the `CGGlyph` value for the specified glyph name in the given font.

`CTFontGetBoundingRectsForGlyphs` (page 31)
Calculates the bounding rects for an array of glyphs and returns the overall bounding rectangle for the glyph run.

`CTFontGetAdvancesForGlyphs` (page 30)
Calculates the advances for an array of glyphs and returns the summed advance.

`CTFontGetVerticalTranslationsForGlyphs` (page 38)
Calculates the offset from the default (horizontal) origin to the vertical origin for an array of glyphs.

Working With Font Variations

`CTFontCopyVariationAxes` (page 22)
Returns an array of variation axes.

`CTFontCopyVariation` (page 22)
Returns a variation dictionary from the font reference.

Getting Font Features

`CTFontCopyFeatures` (page 17)
Returns an array of font features.

`CTFontCopyFeatureSettings` (page 17)
Returns an array of font feature-setting tuples.

Converting Fonts

`CTFontCopyGraphicsFont` (page 18)
Returns a Core Graphics font reference and attributes.

`CTFontCreateWithGraphicsFont` (page 28)
Creates a new font reference from an existing Core Graphics font reference.

Getting Font Table Data

`CTFontCopyAvailableTables` (page 15)
Returns an array of font table tags.

`CTFontCopyTable` (page 21)
Returns a reference to the font table data.

Getting the Type Identifier

`CTFontGetTypeID` (page 37)
Returns the type identifier for Core Text font references.

Functions

CTFontCopyAttribute

Returns the value associated with an arbitrary attribute of the given font.

```
CFTypeRef CTFontCopyAttribute (
   CTFontRef font,
   CFStringRef attribute
);
```

Parameters

font
The font reference.

attribute
The requested attribute.

Return Value
A retained reference to an arbitrary attribute or `NULL` if the requested attribute is not present.

Discussion
Refer to the attribute definitions documentation for information as to how each attribute is packaged as a `CFType`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyAvailableTables

Returns an array of font table tags.

```
CFArrayRef CTFontCopyAvailableTables (
   CTFontRef font,
   CTFontTableOptions options
);
```

Parameters

font
The font reference.

options
The font table options.

Return Value
An array of `Font Table Tag Constants` (page 48) values for the given font and the supplied options.

Discussion
The returned set will contain unboxed values, which can be extracted like so:

```
CTFontTableTag tag = (CTFontTableTag)(uintptr_t)CFArrayGetValueAtIndex(tags,
index);
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyCharacterSet

Returns the Unicode character set of the font.

```
CFCharacterSetRef CTFontCopyCharacterSet (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
A retained reference to the font's character set.

Discussion
The returned character set covers the nominal referenced by the font's Unicode `'cmap'` table.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyDisplayName

Returns the display name of the given font.

```
CFStringRef CTFontCopyDisplayName (
   CTFontRef font
);
```

Parameters

font

The font reference.

Discussion
A retained reference to the localized display name of the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyFamilyName

Returns the family name of the given font.

```
CFStringRef CTFontCopyFamilyName (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
A retained reference to the family name of the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyFeatures

Returns an array of font features.

```
CFArrayRef CTFontCopyFeatures (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
An array of font feature dictionaries for the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyFeatureSettings

Returns an array of font feature-setting tuples.

```
CFArrayRef CTFontCopyFeatureSettings (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
A normalized array of font feature-setting dictionaries. The array contains only the nondefault settings that should be applied to the font, or `NULL` if the default settings should be used.

Discussion
A feature-setting dictionary is a tuple of a `kCTFontFeatureTypeIdentifierKey` (page 43) key-value pair and a `kCTFontFeatureSelectorIdentifierKey` (page 44) key-value pair. Each setting dictionary indicates which setting is enabled. It is the caller's responsibility to handle exclusive and nonexclusive settings as necessary.

The feature settings are verified against those that the font supports and any that do not apply are removed. Further, feature settings that represent a default setting for the font are also removed.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyFontDescriptor

Returns the normalized font descriptor for the given font reference.

```
CTFontDescriptorRef CTFontCopyFontDescriptor (
   CTFontRef font
);
```

Parameters

font
The font reference.

Return Value
A normalized font descriptor for a font containing enough information to recreate this font at a later time.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyFullName

Returns the full name of the given font.

```
CFStringRef CTFontCopyFullName (
   CTFontRef font
);
```

Parameters

font
The font reference.

Return Value
A retained reference to the full name of the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyGraphicsFont

Returns a Core Graphics font reference and attributes.

```
CGFontRef CTFontCopyGraphicsFont (
   CTFontRef font,
   CTFontDescriptorRef *attributes
);
```

Parameters

font

The font reference.

attributes

On output, points to a font descriptor containing additional attributes from the font. Can be `NULL`. Must be released by the caller.

Return Value

A `CGFontRef` object for the given font reference.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFont.h`

CTFontCopyLocalizedName

Returns a reference to a localized name for the given font.

```
CFStringRef CTFontCopyLocalizedName (
   CTFontRef font,
   CFStringRef nameKey,
   CFStringRef *language
);
```

Parameters

font

The font reference.

nameKey

The name specifier. See "Name Specifier Constants" (page 40) for possible values.

language

On output, points to the language string of the returned name string. The format of the language identifier conforms to the RFC 3066bis standard.

Return Value

A specific localized name from the font reference or `NULL` if the font does not have an entry for the requested name key.

Discussion

The name is localized based on the user's global language preference precedence. That is, the user's language preference is a list of languages in order of precedence. So, for example, if the list had Japanese and English, in that order, then a font that did not have Japanese name strings but had English strings would return the English strings.

Availability

Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyName

Returns a reference to the requested name of the given font.

```
CFStringRef CTFontCopyName (
   CTFontRef font,
   CFStringRef nameKey
);
```

Parameters

font

The font reference.

nameKey

The name specifier. See "Name Specifier Constants" (page 40) for possible values.

Return Value
The requested name for the font, or `NULL` if the font does not have an entry for the requested name. The Unicode version of the name is preferred, otherwise the first available version is returned.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyPostScriptName

Returns the PostScript name of the given font.

```
CFStringRef CTFontCopyPostScriptName (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
A retained reference to the PostScript name of the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopySupportedLanguages

Returns an array of languages supported by the font.

```
CFArrayRef CTFontCopySupportedLanguages (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value

A retained reference to an array of languages supported by the font. The array contains language identifier strings as `CFStringRef` objects. The format of the language identifier conforms to the RFC 3066bis standard.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFont.h`

CTFontCopyTable

Returns a reference to the font table data.

```
CFDataRef CTFontCopyTable (
   CTFontRef font,
   CTFontTableTag table,
   CTFontTableOptions options
);
```

Parameters

font

The font reference.

table

The font table identifier as a `Font Table Tag Constants` (page 48) constant. See "Font Table Tag Constants" (page 48) for possible values.

options

The font table options.

Return Value

A retained reference to the font table data as a `CFDataRef` object. The table data is not actually copied; however, the data reference must be released.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFont.h`

CTFontCopyTraits

Returns the traits dictionary of the given font.

```
CFDictionaryRef CTFontCopyTraits (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
A retained reference to the font traits dictionary. Individual traits can be accessed with the trait key constants.

Discussion
See the Constants section of *CTFontDescriptor Reference* for a definition of the font traits.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyVariation

Returns a variation dictionary from the font reference.

```
CFDictionaryRef CTFontCopyVariation (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The current variation instance as a dictionary.

Discussion
The keys for each variation correspond to the variation identifier obtained via `kCTFontVariationAxisIdentifierKey` (page 42), which represents the four-character axis code as a CFNumber object.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCopyVariationAxes

Returns an array of variation axes.

```
CFArrayRef CTFontCopyVariationAxes (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value

An array of variation axes dictionaries. Each variation axis dictionary contains the five variation axis keys listed in "Font Variation Axis Dictionary Keys" (page 42).

Availability

Available in iOS 3.2 and later.

Declared In

CTFont.h

CTFontCreateCopyWithAttributes

Returns a new font with additional attributes based on the original font.

```
CTFontRef CTFontCreateCopyWithAttributes (
   CTFontRef font,
   CGFloat size,
   const CGAffineTransform *matrix,
   CTFontDescriptorRef attributes
);
```

Parameters

font

The original font reference on which to base the new font.

size

The point size for the font reference. If `0.0` is specified, the original font's size is preserved.

matrix

The transformation matrix for the font. If `NULL` is specified, the original font's matrix is preserved.

attributes

A font descriptor containing additional attributes that the new font should contain.

Return Value

A new font reference converted from the original with the specified attributes.

Discussion

This function provides a mechanism to change attributes quickly on a given font reference in response to user actions. For instance, the size can be changed in response to a user manipulating a size slider.

Availability

Available in iOS 3.2 and later.

Declared In

CTFont.h

CTFontCreateCopyWithFamily

Returns a new font in the specified family based on the traits of the original font.

```
CTFontRef CTFontCreateCopyWithFamily (
   CTFontRef font,
   CGFloat size,
   const CGAffineTransform *matrix,
   CFStringRef family
);
```

Parameters

font

The original font reference on which to base the new font.

size

The point size for the font reference. If `0.0` is specified, the original font's size is preserved.

matrix

The transformation matrix for the font. If `NULL` is specified, the original font's matrix is preserved.

family

The name of the desired family.

Return Value

A new font reference with the original traits in the given family, or `NULL` if none is found in the system.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFont.h`

CTFontCreateCopyWithSymbolicTraits

Returns a new font in the same font family as the original with the specified symbolic traits.

```
CTFontRef CTFontCreateCopyWithSymbolicTraits (
   CTFontRef font,
   CGFloat size,
   const CGAffineTransform *matrix,
   CTFontSymbolicTraits symTraitValue,
   CTFontSymbolicTraits symTraitMask
);
```

Parameters

font

The original font reference on which to base the new font.

size

The point size for the font reference. If `0.0` is specified, the original font's size is preserved.

matrix

The transformation matrix for the font. If `NULL` is specified, the original font's matrix is preserved.

symTraitValue

The value of the symbolic traits.

symTraitMask
The mask bits of the symbolic traits.

Return Value
A new font reference in the same family with the given symbolic traits. or `NULL` if none is found in the system.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateForString

Returns a new font reference that can best map the given string range based on the current font.

```
CTFontRef CTFontCreateForString (
   CTFontRef currentFont,
   CFStringRef string,
   CFRange range
);
```

Parameters
currentFont
The current font that contains a valid cascade list.

string
A unicode string containing characters that cannot be encoded by the current font.

range
A `CFRange` structure specifying the range of the string that needs to be mapped.

Return Value
The best substitute font from the cascade list of the current font that can encode the specified string range. If the current font is capable of encoding the string range, then it is retained and returned.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreatePathForGlyph

Creates a path for the specified glyph.

```
CGPathRef CTFontCreatePathForGlyph (
   CTFontRef font,
   CGGlyph glyph,
   const CGAffineTransform *transform
);
```

Parameters
font
The font reference.

glyph
The glyph.

transform
An affine transform applied to the path. Can be `NULL`. If `NULL`, `CGAffineTransformIdentity` is used.

Return Value
A CGPath object containing the glyph outlines, `NULL` on error. Must be released by caller.

Discussion
Creates a path from the outlines of the glyph for the specified font. The path reflects the font point size, matrix, and transform parameter, applied in that order. The transform parameter is most commonly be used to provide a translation to the desired glyph origin.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateUIFontForLanguage

Returns the special user-interface font for the given language and user-interface type.

```
CTFontRef CTFontCreateUIFontForLanguage (
   CTFontUIFontType uiType,
   CGFloat size,
   CFStringRef language
);
```

Parameters

uiType
A constant specifying the intended user-interface use for the requested font reference. See "Enumerations" (page 44) for possible values.

size
The point size for the font reference. If `0.0` is specified, the default size for the requested user-interface type is used.

language
Language specifier string to select a font for a particular localization. If `NULL` is specified, the current system language is used. The format of the language identifier should conform to the RFC 3066bis standard.

Return Value
The correct font for various user-interface uses.

Discussion
The only required parameter is the *uiType* selector; the other parameters have default values.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateWithFontDescriptor

Returns a new font reference that best matches the given font descriptor.

```
CTFontRef CTFontCreateWithFontDescriptor (
   CTFontDescriptorRef descriptor,
   CGFloat size,
   const CGAffineTransform *matrix
);
```

Parameters

descriptor

A font descriptor containing attributes that specify the requested font.

size

The point size for the font reference. If `0.0` is specified, the default font size of `12.0` is used.

matrix

The transformation matrix for the font. If `NULL` is specified, the identity matrix is used.

Return Value

A CTFontRef that best matches the attributes provided with the font descriptor.

Discussion

The *size* and *matrix* parameters override any specified in the font descriptor unless they are unspecified (`0.0` for *size* and `NULL` for *matrix*). A best match font is always returned, and default values are used for any unspecified parameters.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFont.h`

CTFontCreateWithFontDescriptorAndOptions

Returns a new font reference that best matches the given font descriptor.

```
CTFontRef CTFontCreateWithFontDescriptorAndOptions (
   CTFontDescriptorRef descriptor,
   CGFloat size,
   const CGAffineTransform *matrix,
   CTFontOptions options
);
```

Parameters

descriptor

A font descriptor containing attributes that specify the requested font.

size

The point size for the font reference. If 0.0 is specified, the default font size of 12.0 is used.

matrix

The transformation matrix for the font. If If `NULL` is specified, the identity matrix is used.

options

Options flags. See "Font Option Constants" (page 55) for values.

Return Value
A `CTFontRef` that best matches the attributes provided with the font descriptor.

Discussion
The size and matrix parameters override any specified in the font descriptor, unless they are unspecified (`0.0` for *`size`* and `NULL` for *`matrix`* and *`options`*). A best match font is always returned, and default values are used for any unspecified.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateWithGraphicsFont

Creates a new font reference from an existing Core Graphics font reference.

```
CTFontRef CTFontCreateWithGraphicsFont (
   CGFontRef graphicsFont,
   CGFloat size,
   const CGAffineTransform *matrix,
   CTFontDescriptorRef attributes
);
```

Parameters

`graphicsFont`
A valid Core Graphics font reference.

`size`
The point size for the font reference. If `0.0` is specified the default font size of 12.0 is used.

`matrix`
The transformation matrix for the font. If `NULL`, the identity matrix is used. Optional.

`attributes`
Additional attributes that should be matched. Optional.

Return Value
A new font reference for an existing `CGFontRef` object with the specified size, matrix, and additional attributes.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateWithName

Returns a new font reference for the given name.

```
CTFontRef CTFontCreateWithName (
   CFStringRef name,
   CGFloat size,
   const CGAffineTransform *matrix
);
```

Parameters

name

The font name for which you wish to create a new font reference. A valid PostScript name is preferred, although other font name types are matched in a fallback manner.

size

The point size for the font reference. If `0.0` is specified, the default font size of `12.0` is used.

matrix

The transformation matrix for the font. If `NULL` is specified, the identity matrix is used.

Return Value
Returns a `CTFontRef` that best matches the name provided with size and matrix attributes.

Discussion
The *name* parameter is the only required parameter, and default values are used for unspecified parameters (`0.0` for *size* and `NULL` for *matrix*). If all parameters cannot be matched identically, a best match is found.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontCreateWithNameAndOptions

Returns a new font reference for the given name.

```
CTFontRef CTFontCreateWithNameAndOptions (
   CFStringRef name,
   CGFloat size,
   const CGAffineTransform *matrix,
   CTFontOptions options
);
```

Parameters

name

The font name for which you wish to create a new font reference. A valid PostScript name is preferred, although other font name types are matched in a fallback manner.

size

The point size for the font reference. If 0.0 is specified, the default font size of 12.0 is used.

matrix

The transformation matrix for the font. If `NULL` is specified, the identity matrix is used.

options

Options flags. See "Font Option Constants" (page 55) for values.

Return Value
Returns a `CTFontRef` that best matches the name provided with size and matrix attributes.

Discussion
The *name* parameter is the only required parameter, and default values are used for unspecified parameters (`0.0` for *size* and `NULL` for *matrix* and *options*). If all parameters cannot be matched identically, a best match is found.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetAdvancesForGlyphs

Calculates the advances for an array of glyphs and returns the summed advance.

```
double CTFontGetAdvancesForGlyphs (
   CTFontRef font,
   CTFontOrientation orientation,
   const CGGlyph glyphs[],
   CGSize advances[],
   CFIndex count
);
```

Parameters

font
The font reference.

orientation
The intended drawing orientation of the glyphs. Used to determined which glyph metrics to return.

glyphs
An array of `count` number of glyphs.

advances
An array of `count` number of `CGSize` objects to receive the computed glyph advances. If `NULL`, only the overall advance is calculated.

count
The capacity of the *glyphs* and *advances* buffers.

Return Value
The summed glyph advance of an array of glyphs.

Discussion
Individual glyph advances are passed back via the *advances* parameter. These are the ideal metrics for each glyph scaled and transformed in font space.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetAscent

Returns the scaled font-ascent metric of the given font.

```
CGFloat CTFontGetAscent (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The font-ascent metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetBoundingBox

Returns the scaled bounding box of the given font.

```
CGRect CTFontGetBoundingBox (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The design bounding box of the font, which is the rectangle defined by `xMin`, `yMin`, `xMax`, and `yMax` values for the font. Returns `CGRectNull` on error.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetBoundingRectsForGlyphs

Calculates the bounding rects for an array of glyphs and returns the overall bounding rectangle for the glyph run.

```
CGRect CTFontGetBoundingRectsForGlyphs (
   CTFontRef font,
   CTFontOrientation orientation,
   const CGGlyph glyphs[],
   CGRect boundingRects[],
   CFIndex count
);
```

Parameters

font

The font reference.

orientation
The intended drawing orientation of the glyphs. Used to determined which glyph metrics to return.

glyphs
An array of *count* number of glyphs.

boundingRects
On output, the computed glyph rectangles in an array of *count* number of `CGRect` objects. If `NULL`, only the overall bounding rectangle is calculated.

count
The capacity of the *glyphs* and *boundingRects* buffers.

Return Value
The overall bounding rectangle for an array or run of glyphs. Returns `CGRectNull` on error.

Discussion
The bounding rectangles of the individual glyphs are returned through the *boundingRects* parameter. These are the design metrics from the font transformed in font space.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetCapHeight

Returns the cap-height metric of the given font.

```
CGFloat CTFontGetCapHeight (
   CTFontRef font
);
```

Parameters

font
The font reference.

Return Value
The font cap-height metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetDescent

Returns the scaled font-descent metric of the given font.

```
CGFloat CTFontGetDescent (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The font-descent metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetGlyphCount

Returns the number of glyphs of the given font.

```
CFIndex CTFontGetGlyphCount (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The number of glyphs in the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetGlyphsForCharacters

Provides basic Unicode encoding for the given font, returning by reference an array of `CGGlyph` values corresponding to a given array of Unicode characters for the given font.

```
bool CTFontGetGlyphsForCharacters (
   CTFontRef font,
   const UniChar characters[],
   CGGlyph glyphs[],
   CFIndex count
);
```

Parameters

font

The font reference.

characters
An array of Unicode characters.

glyphs
On output, points to an array of glyph values.

count
The capacity of the character and glyph arrays.

Return Value
`True` if the font could encode all Unicode characters; otherwise `False`.

Discussion
If a glyph could not be encoded, a value of `0` is passed back at the corresponding index in the *glyphs* array and the function returns `False`. It is the responsibility of the caller to handle the Unicode properties of the input characters.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetGlyphWithName

Returns the `CGGlyph` value for the specified glyph name in the given font.

```
CGGlyph CTFontGetGlyphWithName (
   CTFontRef font,
   CFStringRef glyphName
);
```

Parameters

font
The font reference.

glyphName
The glyph name as a `CFString` object.

Return Value
The glyph value for the named glyph as a `CGGlyph` object, or if the glyph name is not recognized, the `.notdef` glyph index value.

Discussion
The returned `CGGlyph` object can be used with any of the subsequent glyph data accessors or directly with Core Graphics.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetLeading

Returns the scaled font-leading metric of the given font.

```
CGFloat CTFontGetLeading (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The font-leading metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
CTFont.h

CTFontGetMatrix

Returns the transformation matrix of the given font.

```
CGAffineTransform CTFontGetMatrix (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The transformation matrix for the given font reference. This is the matrix that was provided when the font was created.

Availability
Available in iOS 3.2 and later.

Declared In
CTFont.h

CTFontGetSize

Returns the point size of the given font.

```
CGFloat CTFontGetSize (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The point size of the given font reference. This is the point size provided when the font was created.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetSlantAngle

Returns the slant angle of the given font.

```
CGFloat CTFontGetSlantAngle (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The transformed slant angle of the font. This is equivalent to the italic or caret angle with any skew from the tranformation matrix applied.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetStringEncoding

Returns the best string encoding for legacy format support.

```
CFStringEncoding CTFontGetStringEncoding (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The best string encoding for the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetSymbolicTraits

Returns the symbolic traits of the given font.

```
CTFontSymbolicTraits CTFontGetSymbolicTraits (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The symbolic traits of the font. This is equivalent to the `kCTFontSymbolicTrait` value of the traits dictionary.

Discussion
See the Constants section of *CTFontDescriptor Reference* for a definition of the font traits.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetTypeID

Returns the type identifier for Core Text font references.

```
CFTypeID CTFontGetTypeID (
   void
);
```

Return Value
The identifier for the CTFont opaque type.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetUnderlinePosition

Returns the scaled underline position of the given font.

```
CGFloat CTFontGetUnderlinePosition (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The font underline-position metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetUnderlineThickness

Returns the scaled underline-thickness metric of the given font.

```
CGFloat CTFontGetUnderlineThickness (
   CTFontRef font
);
```

Parameters

font
The font reference.

Return Value
The font underline-thickness metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetUnitsPerEm

Returns the units-per-em metric of the given font.

```
unsigned CTFontGetUnitsPerEm (
   CTFontRef font
);
```

Parameters

font
The font reference.

Return Value
The units per em of the font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetVerticalTranslationsForGlyphs

Calculates the offset from the default (horizontal) origin to the vertical origin for an array of glyphs.

```
void CTFontGetVerticalTranslationsForGlyphs (
   CTFontRef font,
   const CGGlyph glyphs[],
   CGSize translations[],
   CFIndex count
);
```

Parameters

font

The font reference.

glyphs

An array of *count* number of glyphs.

translations

On output, the computed origin offsets in an array of *count* number of `CGSize` objects.

count

The capacity of the *glyphs* and *translations* buffers.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

CTFontGetXHeight

Returns the x-height metric of the given font.

```
CGFloat CTFontGetXHeight (
   CTFontRef font
);
```

Parameters

font

The font reference.

Return Value
The font x-height metric scaled according to the point size and matrix of the font reference.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

Data Types

CTFontRef

A reference to a Core Text font object.

```
typedef const struct __CTFont *CTFontRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFont.h`

Constants

Global Variables

Name Specifier Constants

Name specifier constants provide access to the different names associated with a font.

```
const CFStringRef kCTFontCopyrightNameKey;
const CFStringRef kCTFontFamilyNameKey;
const CFStringRef kCTFontSubFamilyNameKey;
const CFStringRef kCTFontStyleNameKey;
const CFStringRef kCTFontUniqueNameKey;
const CFStringRef kCTFontFullNameKey;
const CFStringRef kCTFontVersionNameKey;
const CFStringRef kCTFontPostScriptNameKey;
const CFStringRef kCTFontTrademarkNameKey;
const CFStringRef kCTFontManufacturerNameKey;
const CFStringRef kCTFontDesignerNameKey;
const CFStringRef kCTFontDescriptionNameKey;
const CFStringRef kCTFontVendorURLNameKey;
const CFStringRef kCTFontDesignerURLNameKey;
const CFStringRef kCTFontLicenseNameKey;
const CFStringRef kCTFontLicenseURLNameKey;
const CFStringRef kCTFontSampleTextNameKey;
const CFStringRef kCTFontPostScriptCIDNameKey;
```

Constants

`kCTFontCopyrightNameKey`
The name specifier for the copyright name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontFamilyNameKey`
The name specifier for the family name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontSubFamilyNameKey`
The name specifier for the subfamily name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontStyleNameKey`
The name specifier for the style name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontUniqueNameKey`
The name specifier for the unique name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontFullNameKey`
The name specifier for the full name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontVersionNameKey`
The name specifier for the version name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontPostScriptNameKey`
The name specifier for the PostScript name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTrademarkNameKey`
The name specifier for the trademark name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontManufacturerNameKey`
The name specifier for the manufacturer name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontDesignerNameKey`
The name specifier for the designer name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontDescriptionNameKey`
The name specifier for the description name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontVendorURLNameKey`
The name specifier for the vendor URL name.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontDesignerURLNameKey`
The name specifier for the designer URL name.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontLicenseNameKey`
The name specifier for the license name.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontLicenseURLNameKey`
The name specifier for the license URL name.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontSampleTextNameKey`
The name specifier for the sample text name string.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontPostScriptCIDNameKey`
The name specifier for the PostScript character identifier (CID) font name.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

Declared In
`CTFont.h`

Font Variation Axis Dictionary Keys

These constants provide keys to font variation axis dictionary values.

```
const CFStringRef kCTFontVariationAxisIdentifierKey;
const CFStringRef kCTFontVariationAxisMinimumValueKey;
const CFStringRef kCTFontVariationAxisMaximumValueKey;
const CFStringRef kCTFontVariationAxisDefaultValueKey;
const CFStringRef kCTFontVariationAxisNameKey;
```

Constants
`kCTFontVariationAxisIdentifierKey`
Key to get the variation axis identifier value as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisMinimumValueKey`
Key to get the variation axis minimum value as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisMaximumValueKey`
Key to get the variation axis maximum value as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisDefaultValueKey`
Key to get the variation axis default value as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisNameKey`
Key to get the localized variation axis name string.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

Declared In
`CTFont.h`

Font Feature Constants

These constants provide keys to font feature dictionary values.

```
const CFStringRef kCTFontFeatureTypeIdentifierKey;
const CFStringRef kCTFontFeatureTypeNameKey;
const CFStringRef kCTFontFeatureTypeExclusiveKey;
const CFStringRef kCTFontFeatureTypeSelectorsKey;
const CFStringRef kCTFontFeatureSelectorIdentifierKey;
const CFStringRef kCTFontFeatureSelectorNameKey;
const CFStringRef kCTFontFeatureSelectorDefaultKey;
const CFStringRef kCTFontFeatureSelectorSettingKey;
```

Constants
`kCTFontFeatureTypeIdentifierKey`
Key to get the font feature type value as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontFeatureTypeNameKey`
Key to get the localized font feature type name as a `CFString` object.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontFeatureTypeExclusiveKey`
Key to get the font feature exclusive setting of the feature as a `CFBoolean` object. The value associated with this key indicates whether the feature selectors associated with this type should be mutually exclusive.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontFeatureTypeSelectorsKey`
: Key to get the the array of font feature selectors as a `CFArrayRef` object. This is an array of selector dictionaries that contain the values for the font feature selector keys listed in this group.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontFeatureSelectorIdentifierKey`
: Key to be used with a selector dictionary corresponding to a feature type to obtain the selector identifier value as a `CFNumberRef` object.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontFeatureSelectorNameKey`
: Key to be used with a selector dictionary to get the localized name string for the selector as a `CFStringRef` object.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontFeatureSelectorDefaultKey`
: Key to be used with a selector dictionary to get the default indicator for the selector. This value is a `CFBooleanRef` object, which if present and true, indicates that this selector is the default setting for the current feature type.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontFeatureSelectorSettingKey`
: Key to be used with a selector dictionary to get or specify the current setting for the selector. This value is a `CFBooleanRef` object to indicate whether this selector is on or off. If this key is not present, the default setting is used.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

Declared In
`CTFont.h`

Enumerations

User Interface Type Constants

These constants represent the specific user-interface purpose to specify for font creation.

```
enum { kCTFontNoFontType = -1,
kCTFontUserFontType = 0,
kCTFontUserFixedPitchFontType = 1,
kCTFontSystemFontType = 2,
kCTFontEmphasizedSystemFontType = 3,
kCTFontSmallSystemFontType = 4,
kCTFontSmallEmphasizedSystemFontType = 5,
kCTFontMiniSystemFontType = 6,
kCTFontMiniEmphasizedSystemFontType = 7,
kCTFontViewsFontType = 8,
kCTFontApplicationFontType = 9,
kCTFontLabelFontType = 10,
kCTFontMenuTitleFontType = 11,
kCTFontMenuItemFontType = 12,
kCTFontMenuItemMarkFontType = 13,
kCTFontMenuItemCmdKeyFontType = 14,
kCTFontWindowTitleFontType = 15,
kCTFontPushButtonFontType = 16,
kCTFontUtilityWindowTitleFontType = 17,
kCTFontAlertHeaderFontType = 18,
kCTFontSystemDetailFontType = 19,
kCTFontEmphasizedSystemDetailFontType = 20,
kCTFontToolbarFontType = 21,
kCTFontSmallToolbarFontType = 22,
kCTFontMessageFontType = 23,
kCTFontPaletteFontType = 24,
kCTFontToolTipFontType = 25,
kCTFontControlContentFontType = 26};
typedef uint32_t CTFontUIFontType;
```

Constants

`kCTFontNoFontType`

The user-interface font type is not specified.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontUserFontType`

The font used by default for documents and other text under the user's control (that is, text whose font the user can normally change).

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontUserFixedPitchFontType`

The font used by default for documents and other text under the user's control when that font is fixed-pitch.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontSystemFontType`

The system font used for standard user-interface items such as button labels, menu items, and so on.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontEmphasizedSystemFontType`
The system font used for emphasis in alerts.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontSmallSystemFontType`
The standard small system font used for informative text in alerts, column headings in lists, help tags, and small controls.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontSmallEmphasizedSystemFontType`
The small system font used for emphasis.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontMiniSystemFontType`
The standard miniature system font used for mini controls and utility window labels and text.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontMiniEmphasizedSystemFontType`
The miniature system font used for emphasis.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontViewsFontType`
The view font used as the default font of text in lists and tables.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontApplicationFontType`
The default font for text documents.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontLabelFontType`
The font used for labels and tick marks on full-size sliders.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontMenuTitleFontType`
The font used for menu titles.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemFontType`
The font used for menu items.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemMarkFontType`
: The font used to draw menu item marks.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontMenuItemCmdKeyFontType`
: The font used for menu-item command-key equivalents.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontWindowTitleFontType`
: The font used for window titles.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontPushButtonFontType`
: The font used for a push button (a rounded rectangular button with a text label on it).

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontUtilityWindowTitleFontType`
: The font used for utility window titles.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontAlertHeaderFontType`
: The font used for alert headers.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontSystemDetailFontType`
: The standard system font used for details.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontEmphasizedSystemDetailFontType`
: The system font used for emphasis in details.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontToolbarFontType`
: The font used for labels of toolbar items.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontSmallToolbarFontType`
: The small font used for labels of toolbar items.

  Available in iOS 3.2 and later.

  Declared in `CTFont.h`.

`kCTFontMessageFontType`
The font used for standard interface items, such as button labels, menu items, and so on.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontPaletteFontType`
The font used in tool palettes.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontToolTipFontType`
The font used for tool tips.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontControlContentFontType`
The font used for contents of user-interface controls.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

Discussion
Use these constants with the `CTFontCreateUIFontForLanguage` (page 26) function to indicate the intended user interface use of the font reference to be created.

Declared In
`CTFont.h`

Font Table Tag Constants

Font table tags provide access to font table data.

```
enum {
kCTFontTableBASE    = 'BASE',
kCTFontTableCFF     = 'CFF ',
kCTFontTableDSIG    = 'DSIG',
kCTFontTableEBDT    = 'EBDT',
kCTFontTableEBLC    = 'EBLC',
kCTFontTableEBSC    = 'EBSC',
kCTFontTableGDEF    = 'GDEF',
kCTFontTableGPOS    = 'GPOS',
kCTFontTableGSUB    = 'GSUB',
kCTFontTableJSTF    = 'JSTF',
kCTFontTableLTSH    = 'LTSH',
kCTFontTableOS2     = 'OS/2',
kCTFontTablePCLT    = 'PCLT',
kCTFontTableVDMX    = 'VDMX',
kCTFontTableVORG    = 'VORG',
kCTFontTableZapf    = 'Zapf',
kCTFontTableAcnt    = 'acnt',
kCTFontTableAvar    = 'avar',
kCTFontTableBdat    = 'bdat',
kCTFontTableBhed    = 'bhed',
kCTFontTableBloc    = 'bloc',
kCTFontTableBsln    = 'bsln',
kCTFontTableCmap    = 'cmap',
kCTFontTableCvar    = 'cvar',
kCTFontTableCvt     = 'cvt ',
kCTFontTableFdsc    = 'fdsc',
kCTFontTableFeat    = 'feat',
kCTFontTableFmtx    = 'fmtx',
kCTFontTableFpgm    = 'fpgm',
kCTFontTableFvar    = 'fvar',
kCTFontTableGasp    = 'gasp',
kCTFontTableGlyf    = 'glyf',
kCTFontTableGvar    = 'gvar',
kCTFontTableHdmx    = 'hdmx',
kCTFontTableHead    = 'head',
kCTFontTableHhea    = 'hhea',
kCTFontTableHmtx    = 'hmtx',
kCTFontTableHsty    = 'hsty',
kCTFontTableJust    = 'just',
kCTFontTableKern    = 'kern',
kCTFontTableLcar    = 'lcar',
kCTFontTableLoca    = 'loca',
kCTFontTableMaxp    = 'maxp',
kCTFontTableMort    = 'mort',
kCTFontTableMorx    = 'morx',
kCTFontTableName    = 'name',
kCTFontTableOpbd    = 'opbd',
kCTFontTablePost    = 'post',
kCTFontTablePrep    = 'prep',
kCTFontTableProp    = 'prop',
kCTFontTableTrak    = 'trak',
kCTFontTableVhea    = 'vhea',
kCTFontTableVmtx    = 'vmtx'
};
typedef uint32_t CTFontTableTag;
```

Constants

`kCTFontTableBASE`
: Font table tag for the font baseline.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableCFF`
: Font table tag for a PostScript font program.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableDSIG`
: Font table tag for a digital signature.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableEBDT`
: Font table tag for an embedded bitmap.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableEBLC`
: Font table tag for the embedded bitmap location.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableEBSC`
: Font table tag for embedded bitmap scaling.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableGDEF`
: Font table tag for glyph definition.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableGPOS`
: Font table tag for glyph positioning.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableGSUB`
: Font table tag for glyph substitution.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableJSTF`
: Font table tag for justification.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableLTSH`
Font table tag for linear threshold.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableOS2`
Font table tag for OS/2 and Windows-specific metrics.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTablePCLT`
Font table tag for PCL 5 data.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableVDMX`
Font table tag for vertical device metrics.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableVORG`
Font table tag for vertical origin.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableZapf`
Font table tag for glyph reference.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableAcnt`
Font table tag for accent attachment.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableAvar`
Font table tag for axis variation.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableBdat`
Font table tag for bitmap data.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableBhed`
Font table tag for bitmap font header.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableBloc`
Font table tag for bitmap location.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableBsln`
Font table tag for baseline.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableCmap`
Font table tag for character-to-glyph mapping.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableCvar`
Font table tag for control value variation, or CVT variation.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableCvt`
Font table tag for control value table.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableFdsc`
Font table tag for font descriptor.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableFeat`
Font table tag for layout feature.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableFmtx`
Font table tag for font metrics.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableFpgm`
Font table tag for font program.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableFvar`
Font table tag for font variation.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableGasp`
Font table tag for grid-fitting/scan-conversion.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableGlyf`
Font table tag for glyph data.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableGvar`
Font table tag for glyph variation.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableHdmx`
Font table tag for horizontal device metrics.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableHead`
Font table tag for font header.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableHhea`
Font table tag for horizontal header.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableHmtx`
Font table tag for horizontal metrics.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableHsty`
Font table tag for horizontal style.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableJust`
Font table tag for justification.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableKern`
Font table tag for kerning.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableLcar`
Font table tag for ligature caret.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableLoca`
Font table tag for index to location.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableMaxp`
Font table tag for maximum profile.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableMort`
Font table tag for morph.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableMorx`
Font table tag for extended morph.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableName`
Font table tag for naming table.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableOpbd`
Font table tag for optical bounds.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTablePost`
Font table tag for PostScript information.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTablePrep`
Font table tag for control value program, `'prep'` table.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableProp`
Font table tag for properties.
Available in iOS 3.2 and later.
Declared in `CTFont.h`.

`kCTFontTableTrak`
Font table tag for tracking.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableVhea`
Font table tag for vertical header.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableVmtx`
Font table tag for vertical metrics.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

Declared In
`CTFont.h`

Font Table Option Constants

These constants describe font table options.

```
enum {
kCTFontTableOptionNoOptions = 0,
kCTFontTableOptionExcludeSynthetic = (1 << 0)
};
typedef uint32_t CTFontTableOptions;
```

Constants
`kCTFontTableOptionNoOptions`
No font table options are specified.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontTableOptionExcludeSynthetic`
The font table excludes synthetic font data.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

Declared In
`CTFont.h`

Font Option Constants

These constants describe options for font creation and descriptor matching. They are used by the functions `CTFontCreateWithNameAndOptions` (page 29) and `CTFontCreateWithFontDescriptorAndOptions` (page 27).

```
enum {
    kCTFontOptionsDefault               = 0,
    kCTFontOptionsPreventAutoActivation = 1 << 0,
    kCTFontOptionsPreferSystemFont      = 1 << 2,
};
typedef CFOptionFlags CTFontOptions;
```

Constants

`kCTFontOptionsDefault`

Default options are used.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontOptionsPreventAutoActivation`

Prevents automatic font activation.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

`kCTFontOptionsPreferSystemFont`

Font matching prefers to match Apple system fonts.

Available in iOS 3.2 and later.

Declared in `CTFont.h`.

CTFontCollection Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTFontCollection.h

Overview

The CTFontCollection opaque type represents a font collection, that is, a group of font descriptors taken together as a single object.

Font collections provide the capabilities of font enumeration, access to global and custom font collections, and access to the font descriptors comprising the collection.

Functions by Task

Creating Font Collections

`CTFontCollectionCreateFromAvailableFonts` (page 58)
Returns a new font collection containing all available fonts.

`CTFontCollectionCreateWithFontDescriptors` (page 59)
Returns a new font collection based on the given array of font descriptors.

`CTFontCollectionCreateCopyWithFontDescriptors` (page 58)
Returns a copy of the original collection augmented with the given new font descriptors.

Getting Font Descriptors

`CTFontCollectionCreateMatchingFontDescriptors` (page 59)
Returns an array of font descriptors matching the collection.

`CTFontCollectionCreateMatchingFontDescriptorsSortedWithCallback` (page 59)
Returns the array of matching font descriptors sorted with the callback function.

Getting the Type Identifier

`CTFontCollectionGetTypeID` (page 60)
Returns the type identifier for Core Text font collection references.

Functions

CTFontCollectionCreateCopyWithFontDescriptors

Returns a copy of the original collection augmented with the given new font descriptors.

```
CTFontCollectionRef CTFontCollectionCreateCopyWithFontDescriptors (
   CTFontCollectionRef original,
   CFArrayRef descriptors,
   CFDictionaryRef options
);
```

Parameters

original
The original font collection reference.

descriptors
An array of font descriptors to augment those of the original collection.

options
The options dictionary. For possible values, see "Constants" (page 61).

Return Value

A copy of the original font collection augmented by the new font descriptors and options.

Discussion

The new font descriptors are merged with the existing descriptors to create a single set.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFontCollection.h`

CTFontCollectionCreateFromAvailableFonts

Returns a new font collection containing all available fonts.

```
CTFontCollectionRef CTFontCollectionCreateFromAvailableFonts (
   CFDictionaryRef options
);
```

Parameters

options
The options dictionary. For possible values, see "Constants" (page 61).

Return Value

A new collection containing all fonts available to the current application.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFontCollection.h`

CTFontCollectionCreateMatchingFontDescriptors

Returns an array of font descriptors matching the collection.

```
CFArrayRef CTFontCollectionCreateMatchingFontDescriptors (
   CTFontCollectionRef collection
);
```

Parameters

collection

The font collection reference.

Return Value

A retained reference to an array of normalized font descriptors matching the collection definition.

Availability

Available in iOS 3.2 and later.

Declared In

CTFontCollection.h

CTFontCollectionCreateMatchingFontDescriptorsSortedWithCallback

Returns the array of matching font descriptors sorted with the callback function.

```
CFArrayRef CTFontCollectionCreateMatchingFontDescriptorsSortedWithCallback (
   CTFontCollectionRef collection,
   CTFontCollectionSortDescriptorsCallback sortCallback,
   void *refCon
);
```

Parameters

collection

The collection reference.

sortCallback

The sorting callback function that defines the sort order.

refCon

Pointer to client data define context for the callback.

Return Value

An array of font descriptors matching the criteria of the collection sorted by the results of the sorting callback function.

Availability

Available in iOS 3.2 and later.

Declared In

CTFontCollection.h

CTFontCollectionCreateWithFontDescriptors

Returns a new font collection based on the given array of font descriptors.

```
CTFontCollectionRef CTFontCollectionCreateWithFontDescriptors (
   CFArrayRef descriptors,
   CFDictionaryRef options
);
```

Parameters

descriptors

An array of font descriptors.

options

The options dictionary. For possible values, see "Constants" (page 61).

Return Value
A new font collection based on the provided font descriptors.

Discussion
The contents of the returned collection are defined by matching the provided descriptors against all available font descriptors.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontCollection.h`

CTFontCollectionGetTypeID

Returns the type identifier for Core Text font collection references.

```
CFTypeID CTFontCollectionGetTypeID (
   void
);
```

Return Value
The identifier for the opaque type CTFontCollection.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontCollection.h`

Data Types

CTFontCollectionRef

A reference to a font collection.

```
typedef const struct __CTFontCollection * CTFontCollectionRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontCollection.h`

Constants

kCTFontCollectionRemoveDuplicatesOption

An option key to specify filtering of duplicates.

```
const CFStringRef kCTFontCollectionRemoveDuplicatesOption;
```

Constants
`kCTFontCollectionRemoveDuplicatesOption`
Option key to specify filtering of duplicates.

Available in iOS 3.2 and later.

Declared in `CTFontCollection.h`.

Discussion
Specify this option key in the options dictionary with a nonzero value to enable automatic filtering of duplicate font descriptors.

Declared In
`CTFontCollection.h`

CTFontDescriptor Reference

Derived From:	`CFTypeRef`
Framework:	ApplicationServices/CoreText
Declared in	CTFontDescriptor.h

Overview

The CTFontDescriptor opaque type represents a font descriptor, that is, a dictionary of attributes (such as name, point size, and variation) that can completely specify a font.

A font descriptor can be an incomplete specification, in which case the system chooses the most appropriate font to match the given attributes.

Functions by Task

Creating Font Descriptors

`CTFontDescriptorCreateWithNameAndSize` (page 69)
Creates a new font descriptor with the provided PostScript name and size.

`CTFontDescriptorCreateWithAttributes` (page 68)
Creates a new font descriptor reference from a dictionary of attributes.

`CTFontDescriptorCreateCopyWithAttributes` (page 66)
Creates a copy of the original font descriptor with new attributes.

`CTFontDescriptorCreateCopyWithVariation` (page 67)
Creates a copy of the original font descriptor with a new variation instance.

`CTFontDescriptorCreateCopyWithFeature` (page 66)
Copies a font descriptor with new feature settings.

`CTFontDescriptorCreateMatchingFontDescriptors` (page 68)
Returns an array of normalized font descriptors matching the provided descriptor.

`CTFontDescriptorCreateMatchingFontDescriptor` (page 67)
Returns the single preferred matching font descriptor based on the original descriptor and system precedence.

Getting Attributes

`CTFontDescriptorCopyAttributes` (page 64)
Returns the attributes dictionary of the font descriptor.

`CTFontDescriptorCopyAttribute` (page 64)
Returns the value associated with an arbitrary attribute.

`CTFontDescriptorCopyLocalizedAttribute` (page 65)
Returns a localized value for the requested attribute, if available.

Getting the Font Descriptor Type

`CTFontDescriptorGetTypeID` (page 69)
Returns the type identifier for Core Text font descriptor references.

Functions

CTFontDescriptorCopyAttribute

Returns the value associated with an arbitrary attribute.

```
CFTypeRef CTFontDescriptorCopyAttribute (
   CTFontDescriptorRef descriptor,
   CFStringRef attribute
);
```

Parameters

descriptor
The font descriptor.

attribute
The requested attribute.

Return Value
A retained reference to an arbitrary attribute, or `NULL` if the requested attribute is not present.

Discussion
Refer to "Font Attributes" (page 70) for documentation explaining how each attribute is packaged as a CFType object.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

CTFontDescriptorCopyAttributes

Returns the attributes dictionary of the font descriptor.

```
CFDictionaryRef CTFontDescriptorCopyAttributes (
   CTFontDescriptorRef descriptor
);
```

Parameters

descriptor

The font descriptor.

Return Value

The font descriptor attributes dictionary. This dictionary contains the minimum number of attributes to specify fully this particular font descriptor.

Availability

Available in iOS 3.2 and later.

Declared In

CTFontDescriptor.h

CTFontDescriptorCopyLocalizedAttribute

Returns a localized value for the requested attribute, if available.

```
CFTypeRef CTFontDescriptorCopyLocalizedAttribute (
   CTFontDescriptorRef descriptor,
   CFStringRef attribute,
   CFStringRef *language
);
```

Parameters

descriptor

The font descriptor.

attribute

The requested font attribute.

language

On output, contains a reference to the matched language. The language identifier will conform to the RFC 3066bis standard.

Return Value

A retained reference to a localized attribute based on the global language list.

Discussion

This function passes back the matched language in *language*. If localization is not possible for the attribute, the behavior matches the value returned from `CTFontDescriptorCopyAttribute` (page 64). Generally, localization of attributes is applicable to name attributes of only a normalized font descriptor.

Availability

Available in iOS 3.2 and later.

Declared In

CTFontDescriptor.h

CTFontDescriptorCreateCopyWithAttributes

Creates a copy of the original font descriptor with new attributes.

```
CTFontDescriptorRef CTFontDescriptorCreateCopyWithAttributes (
   CTFontDescriptorRef original,
   CFDictionaryRef attributes
);
```

Parameters

original

The original font descriptor.

attributes

A dictionary containing arbitrary attributes.

Return Value

A new copy of the original font descriptor with attributes augmented by those specified. If there are conflicts between attributes, the new attributes replace existing ones.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFontDescriptor.h`

CTFontDescriptorCreateCopyWithFeature

Copies a font descriptor with new feature settings.

```
CTFontDescriptorRef CTFontDescriptorCreateCopyWithFeature (
   CTFontDescriptorRef original,
   CFNumberRef featureTypeIdentifier,
   CFNumberRef featureSelectorIdentifier
);
```

Parameters

original

The original font descriptor.

featureTypeIdentifier

The feature type identifier.

featureSelectorIdentifier

The feature selector identifier.

Return Value

A copy of the original font descriptor modified with the given feature settings.

Discussion

This is a convenience method to toggle more easily the state of individual features.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFontDescriptor.h`

CTFontDescriptorCreateCopyWithVariation

Creates a copy of the original font descriptor with a new variation instance.

```
CTFontDescriptorRef CTFontDescriptorCreateCopyWithVariation (
   CTFontDescriptorRef original,
   CFNumberRef variationIdentifier,
   CGFloat variationValue
);
```

Parameters

original

The original font descriptor.

variationIdentifier

The variation axis identifier. This is the four-character code of the variation axis as a CFNumber object.

variationValue

The value corresponding with the variation instance.

Return Value

A copy of the original font descriptor with a new variation instance.

Discussion

This is a convenience method for easily creating new variation font instances.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFontDescriptor.h`

CTFontDescriptorCreateMatchingFontDescriptor

Returns the single preferred matching font descriptor based on the original descriptor and system precedence.

```
CTFontDescriptorRef CTFontDescriptorCreateMatchingFontDescriptor (
   CTFontDescriptorRef descriptor,
   CFSetRef mandatoryAttributes
);
```

Parameters

descriptor

The original font descriptor.

mandatoryAttributes

A set of attribute keys which must be identically matched in any returned font descriptors. May be `NULL`.

Return Value

A retained, normalized font descriptor matching the attributes present in *descriptor*.

Discussion

The original descriptor may be returned in normalized form. The caller is responsible for releasing the result. In the context of font descriptors, *normalized* infers that the input values were matched up with actual existing fonts, and the descriptors for those existing fonts are the returned normalized descriptors.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

CTFontDescriptorCreateMatchingFontDescriptors

Returns an array of normalized font descriptors matching the provided descriptor.

```
CFArrayRef CTFontDescriptorCreateMatchingFontDescriptors (
   CTFontDescriptorRef descriptor,
   CFSetRef mandatoryAttributes
);
```

Parameters

descriptor
The font descriptor.

mandatoryAttributes
A set of attribute keys that must be identically matched in any returned font descriptors. May be `NULL`.

Return Value
A retained array of normalized font descriptors matching the attributes present in *descriptor*.

Discussion
If *descriptor* itself is normalized, then the array will contain only one item: the original descriptor. In the context of font descriptors, *normalized* infers that the input values were matched up with actual existing fonts, and the descriptors for those existing fonts are the returned normalized descriptors.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

CTFontDescriptorCreateWithAttributes

Creates a new font descriptor reference from a dictionary of attributes.

```
CTFontDescriptorRef CTFontDescriptorCreateWithAttributes (
   CFDictionaryRef attributes
);
```

Parameters

attributes
A dictionary containing arbitrary attributes.

Return Value
A new font descriptor with the attributes specified.

Discussion
The provided attribute dictionary can contain arbitrary attributes that are preserved; however, unrecognized attributes are ignored on font creation and and may not be preserved over the round trip from descriptor to font and back to descriptor.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

CTFontDescriptorCreateWithNameAndSize

Creates a new font descriptor with the provided PostScript name and size.

```
CTFontDescriptorRef CTFontDescriptorCreateWithNameAndSize (
   CFStringRef name,
   CGFloat size
);
```

Parameters

name
The PostScript name to be used for the font descriptor as a `CFStringRef` object.

size
The point size. If `0.0`, the font size attribute (`kCTFontSizeAttribute` (page 72)) is omitted from the returned font descriptor.

Return Value
A new font descriptor reference with the given PostScript name and point size.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

CTFontDescriptorGetTypeID

Returns the type identifier for Core Text font descriptor references.

```
CFTypeID CTFontDescriptorGetTypeID (
   void
);
```

Return Value
The identifier for the CTFontDescriptor opaque type.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

Data Types

CTFontDescriptorRef

A reference to a CTFontDescriptor object.

```
typedef const struct __CTFontDescriptor *CTFontDescriptorRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFontDescriptor.h`

Constants

Font Attributes

Font Attribute Constants

These constants are keys for accessing font attributes from a font descriptor.

```
const CFStringRef kCTFontURLAttribute;
const CFStringRef kCTFontNameAttribute;
const CFStringRef kCTFontDisplayNameAttribute;
const CFStringRef kCTFontFamilyNameAttribute;
const CFStringRef kCTFontStyleNameAttribute;
const CFStringRef kCTFontTraitsAttribute;
const CFStringRef kCTFontVariationAttribute;
const CFStringRef kCTFontSizeAttribute;
const CFStringRef kCTFontMatrixAttribute;
const CFStringRef kCTFontCascadeListAttribute;
const CFStringRef kCTFontCharacterSetAttribute;
const CFStringRef kCTFontLanguagesAttribute;
const CFStringRef kCTFontBaselineAdjustAttribute;
const CFStringRef kCTFontMacintoshEncodingsAttribute;
const CFStringRef kCTFontFeaturesAttribute;
const CFStringRef kCTFontFeatureSettingsAttribute;
const CFStringRef kCTFontFixedAdvanceAttribute;
const CFStringRef kCTFontOrientationAttribute;
const CFStringRef kCTFontFormatAttribute;
const CFStringRef kCTFontRegistrationScopeAttribute;
const CFStringRef kCTFontPriorityAttribute;
const CFStringRef kCTFontEnabledAttribute;
```

Constants

`kCTFontURLAttribute`

Key for accessing the font URL from the font descriptor. The value associated with this key is a `CFURLRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontNameAttribute`

Key for accessing the PostScript name from the font descriptor. The value associated with this key is a `CFStringRef` object. If the value is unspecified, it defaults to `Helvetica`, and if that font is unavailable, it falls back to the global font cascade list.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontDisplayNameAttribute`

Key for accessing the name used to display the font. Most commonly this is the full name. The value associated with this key is a `CFStringRef` object. If the value is unspecified, it defaults to `Helvetica`, and if that font is unavailable, it falls back to the global font cascade list.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFamilyNameAttribute`

Key for accessing the font family name from the font descriptor. The value associated with this key is a `CFStringRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontStyleNameAttribute`

Key for accessing the style name of the font. This name represents the designer's description of the font's style. The value associated with this key is a `CFStringRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontTraitsAttribute`
: Key for accessing the dictionary of font traits for stylistic information. See "Font Traits" (page 77) for the list of font traits. The value associated with this key is a `CFDictionaryRef` object.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontVariationAttribute`
: Key to obtain the font variation dictionary instance as a `CFDictionaryRef` object. If specified in a font descriptor, fonts with the specified axes are primary match candidates; if no such fonts exist, this attribute is ignored.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontSizeAttribute`
: Key to obtain or specify the font point size. Creating a font with this unspecified will default to a point size of 12.0. The value for this key is represented as a `CFNumberRef` object.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontMatrixAttribute`
: Key to specify the font transformation matrix when creating a font. If unspecified it defaults to the unit matrix. The value for this key is a `CFDataRef` object containing a `CGAffineTransform` object.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontCascadeListAttribute`
: Key to specify or obtain the cascade list used for a font reference. The cascade list is a `CFArrayRef` object containing `CTFontDescriptorRef` elements. If unspecified, the global cascade list is used.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontCharacterSetAttribute`
: Key to specify or obtain the Unicode character coverage set for a font reference. The value for this key is a `CFCharacterSetRef` object. If specified, this attribute can be used to restrict the font to a subset of its actual character set. If unspecified, this attribute is ignored and the actual character set is used.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontLanguagesAttribute`
: Key to specify or obtain a list of covered languages for a font reference. The value for this key is a `CFArrayRef` object containing `CFStringRef` elements. If specified, this attribute restricts the search to matching fonts that support the specified languages. The language identifier string should conform to the RFC 3066bis standard. If unspecified, this attribute is ignored.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontBaselineAdjustAttribute`

Key to specify or obtain the baseline adjustment for a font reference. This is primarily used when defining font descriptors for a cascade list to keep the baseline of all fonts even. The value associated with this is a float represented as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontMacintoshEncodingsAttribute`

Key to specify or obtain the Macintosh encodings for a font reference. The value associated with this key is a `CFNumberRef` object containing a bit field of the Macintosh encodings. This attribute is provided for legacy compatibility.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFeaturesAttribute`

Key to specify or obtain the font features for a font reference. The value associated with this key is a `CFArrayRef` object containing font feature dictionaries. This feature list contains the feature information from the `'feat'` table of the font. For more information, see `CTFontCopyFeatures` (page 17).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFeatureSettingsAttribute`

Key to specify or obtain the font features settings for a font reference. The value associated with this key is a `CFArrayRef` object containing font feature-setting dictionaries. A feature-setting dictionary contains a tuple of a `kCTFontFeatureTypeIdentifierKey` (page 43) key-value pair and a `kCTFontFeatureSelectorIdentifierKey` (page 44) key-value pair. Each setting dictionary indicates which setting should be turned on. In the case of duplicate or conflicting setting, the last setting in the list takes precedence. It is the caller's responsibility to handle exclusive and nonexclusive settings as necessary.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFixedAdvanceAttribute`

Key to specify a fixed advance to be used for a font reference. If present and specified, this attribute is used to specify a constant advance to override any font values. The value associated with this key is a float represented as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontOrientationAttribute`

Key to specify a particular orientation for the glyphs of the font. The value associated with this key is an integer represented as a `CFNumberRef` object containing one of the constants in "Font Orientation Constants" (page 74). If you want to receive vertical metrics from a font for vertical rendering, specify `kCTFontVerticalOrientation` (page 75). If unspecified, the font uses its native orientation.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatAttribute`
: Key to specify or obtain the recognized format of the font. The value associated with this key is an integer represented as a `CFNumberRef` object containing one of the constants in "Font Format Constants" (page 75).

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontRegistrationScopeAttribute`
: Key to specify or obtain the font descriptor's registration scope. The value associated with this key is an integer represented as a `CFNumberRef` object containing one of the CTFontManagerScope enumerated values. A value of `NULL` can be returned for font descriptors that are not registered.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontPriorityAttribute`
: Key to specify or obtain the font priority used by font descriptors when resolving duplicates and sorting match results. The value associated with this key is an integer represented as a `CFNumberRef` object containing one of the values enumerated in "Font Priority Constants" (page 76). The higher the value, the higher the priority of the font. Only registered fonts have a priority. Unregistered font descriptors return `NULL`.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontEnabledAttribute`
: Key to obtain the font enabled state. The returned value is an integer represented as a `CFNumberRef` object representing a Boolean value. Unregistered font descriptors return `NULL`, which is equivalent to `false`.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

Font Orientation Constants

Specifies the intended rendering orientation of the font for obtaining glyph metrics. These constants are used as values of `kCTFontOrientationAttribute` (page 73).

```
enum {
     kCTFontDefaultOrientation = 0,
     kCTFontHorizontalOrientation = 1,
     kCTFontVerticalOrientation = 2
};
typedef uint32_t CTFontOrientation;
```

Constants

`kCTFontDefaultOrientation`
: The native orientation of the font.

  Available in iOS 3.2 and later.

  Declared in `CTFontDescriptor.h`.

`kCTFontHorizontalOrientation`
Specifies horizontal orientation.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontVerticalOrientation`
Specifies vertical orientation.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

Font Format Constants

Specifies the recognized format of the font.

```
enum {
    kCTFontFormatUnrecognized       = 0,
    kCTFontFormatOpenTypePostScript = 1,
    kCTFontFormatOpenTypeTrueType   = 2,
    kCTFontFormatTrueType           = 3,
    kCTFontFormatPostScript         = 4,
    kCTFontFormatBitmap             = 5
};
typedef uint32_t CTFontFormat;
```

Constants

`kCTFontFormatUnrecognized`
The font is not a recognized format.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatOpenTypePostScript`
The font is an OpenType format containing PostScript data.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatOpenTypeTrueType`
The font is an OpenType format containing TrueType data.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatTrueType`
The font is a recognized TrueType format.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatPostScript`
The font is a recognized PostScript format.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontFormatBitmap`
The font is a bitmap-only format.

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

Font Priority Constants

Specifies the priority of font descriptors when resolving duplicates and sorting match results.

```
enum {
    kCTFontPrioritySystem =  10000,
    kCTFontPriorityNetwork =  20000,
    kCTFontPriorityComputer =  30000,
    kCTFontPriorityUser =  40000,
    kCTFontPriorityDynamic =  50000,
    kCTFontPriorityProcess =  60000
};
typedef uint32_t CTFontPriority;
```

Constants

`kCTFontPrioritySystem`
Priority of system fonts (located in `/System/Library/Fonts`).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontPriorityNetwork`
Priority of network fonts (located in `/Network/Library/Fonts`).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontPriorityComputer`
Priority of computer local fonts (located in `/Library/Fonts`).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontPriorityUser`
Priority of local fonts (located in user's `Library/Fonts`).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontPriorityDynamic`
Priority of fonts registered dynamically, not located in a standard location (either kCTFontManagerScopeUser or kCTFontManagerScopeSession).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontPriorityProcess`
Priority of fonts registered for the process (kCTFontManagerScopeProcess).

Available in iOS 3.2 and later.

Declared in `CTFontDescriptor.h`.

Font Traits

Font Trait Constants

These constants are keys for accessing font traits from a font descriptor.

```
const CFStringRef kCTFontSymbolicTrait;
const CFStringRef kCTFontWeightTrait;
const CFStringRef kCTFontWidthTrait;
const CFStringRef kCTFontSlantTrait;
```

Constants

`kCTFontSymbolicTrait`

Key to access the symbolic traits value from the font traits dictionary. The value is returned as a `CFNumberRef` object.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontWeightTrait`

Key to access the normalized weight trait from the font traits dictionary. The value returned is a `CFNumberRef` representing a float value between `-1.0` and `1.0` for normalized weight. The value of `0.0` corresponds to the regular or medium font weight.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontWidthTrait`

Key to access the normalized proportion (width condense or expand) trait from the font traits dictionary. This value corresponds to the relative interglyph spacing for a given font. The value returned is a `CFNumberRef` object representing a float between `-1.0` and `1.0`. The value of `0.0` corresponds to regular glyph spacing, and negative values represent condensed glyph spacing.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontSlantTrait`

Key to access the normalized slant angle from the font traits dictionary. The value returned is a `CFNumberRef` object representing a float value between `-1.0` and `1.0` for normalized slant angle. The value of `0.0` corresponds to 0 degrees clockwise rotation from the vertical and `1.0` corresponds to 30 degrees clockwise rotation.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

Font Class Mask Shift Constants

These constants represent the font class mask shift.

```
enum { kCTFontClassMaskShift = 28};
```

Constants

`kCTFontClassMaskShift`

Value used to shift the font class to the uppermost four bits of the symbolic traits

Font Symbolic Traits Constants

These constants represent the symbolic representation of stylistic font attributes.

```
enum {
kCTFontItalicTrait = (1 << 0),
kCTFontBoldTrait = (1 << 1),
kCTFontExpandedTrait = (1 << 5),
kCTFontCondensedTrait = (1 << 6),
kCTFontMonoSpaceTrait = (1 << 10),
kCTFontVerticalTrait = (1 << 11),
kCTFontUIOptimizedTrait = (1 << 12),
kCTFontClassMaskTrait = (15 << kCTFontClassMaskShift)
};
typedef uint32_t CTFontSymbolicTraits;
```

Constants

`kCTFontItalicTrait`
The font typestyle is italic. Additional detail is available via `kCTFontSlantTrait` (page 77).

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontBoldTrait`
The font typestyle is boldface. Additional detail is available via `kCTFontWeightTrait` (page 77).

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontExpandedTrait`
The font typestyle is expanded. Expanded and condensed traits are mutually exclusive.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontCondensedTrait`
The font typestyle is condensed. Expanded and condensed traits are mutually exclusive. Additional detail is available via `kCTFontWidthTrait` (page 77).

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontMonoSpaceTrait`
The font uses fixed-pitch glyphs if available. The font may have multiple glyph advances (many CJK glyphs contain two spaces).

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontVerticalTrait`
The font uses vertical glyph variants and metrics.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontUIOptimizedTrait`
The font synthesizes appropriate attributes for user interface rendering, such as control titles, if necessary.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontClassMaskTrait`
Mask for the font class.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

Discussion
`CTFontSymbolicTraits` symbolically describes stylistic aspects of a font. The upper 16 bits are used to describe appearance of the font, whereas the lower 16 bits are for typeface information. The font appearance information represented by the upper 16 bits can be used for stylistic font matching.

Font Stylistic Class Constants

These constants represent the stylistic class values of the font.

```
enum {
kCTFontUnknownClass = (0 << kCTFontClassMaskShift),
kCTFontOldStyleSerifsClass = (1 << kCTFontClassMaskShift),
kCTFontTransitionalSerifsClass = (2 << kCTFontClassMaskShift),
kCTFontModernSerifsClass = (3 << kCTFontClassMaskShift),
kCTFontClarendonSerifsClass = (4 << kCTFontClassMaskShift),
kCTFontSlabSerifsClass = (5 << kCTFontClassMaskShift),
kCTFontFreeformSerifsClass = (7 << kCTFontClassMaskShift),
kCTFontSansSerifClass = (8 << kCTFontClassMaskShift),
kCTFontOrnamentalsClass = (9 << kCTFontClassMaskShift),
kCTFontScriptsClass = (10 << kCTFontClassMaskShift),
kCTFontSymbolicClass = (12 << kCTFontClassMaskShift)
};
typedef uint32_t CTFontStylisticClass;
```

Constants
`kCTFontUnknownClass`
The font has no design classification.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontOldStyleSerifsClass`
The font's style is based on the Latin printing style of the 15th to 17th century.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontTransitionalSerifsClass`
The font's style is based on the Latin printing style of the 18th to 19th century.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontModernSerifsClass`
The font's style is based on the Latin printing style of the 20th century.

Available in iOS 3.2 and later.

Declared in `CTFontTraits.h`.

`kCTFontClarendonSerifsClass`
: The font's style is a variation of the Oldstyle Serifs and the Transitional Serifs.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontSlabSerifsClass`
: The font's style is characterized by serifs with a square transition between the strokes and the serifs (no brackets).

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontFreeformSerifsClass`
: The font's style includes serifs, but it expresses a design freedom that does not generally fit within the other serif design classifications.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontSansSerifClass`
: The font's style includes most basic letter forms (excluding Scripts and Ornamentals) that do not have serifs on the strokes.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontOrnamentalsClass`
: The font's style includes highly decorated or stylized character shapes such as those typically used in headlines.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontScriptsClass`
: The font's style is among those typefaces designed to simulate handwriting.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

`kCTFontSymbolicClass`
: The font's style is generally design independent, making it suitable for special characters (icons, dingbats, technical symbols, and so on) that may be used equally well with any font.

  Available in iOS 3.2 and later.

  Declared in `CTFontTraits.h`.

Discussion

`CTFontStylisticClass` classifies certain stylistic qualities of the font. These values correspond closely to the font class values in the OpenType OS/2 table. The class values are bundled in the upper four bits of the "`Font Symbolic Traits Constants`" (page 78) and can be obtained via `kCTFontClassMaskTrait` (page 79).

CTFrame Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTFrame.h

Overview

The CTFrame opaque type represents a frame containing multiple lines of text. The frame object is the output resulting from the text-framing process performed by a framesetter object.

You can draw the entire text frame directly into the current graphic context. The frame object contains an array of line objects that can be retrieved for individual rendering or to get glyph information.

Functions by Task

Getting Frame Data

`CTFrameGetStringRange` (page 84)
Returns the range of characters originally requested to fill the frame.

`CTFrameGetVisibleStringRange` (page 85)
Returns the range of characters that actually fit in the frame.

`CTFrameGetPath` (page 84)
Returns the path used to create the frame.

`CTFrameGetFrameAttributes` (page 82)
Returns the frame attributes used to create the frame.

Getting Lines

`CTFrameGetLines` (page 83)
Returns an array of lines stored in the frame.

`CTFrameGetLineOrigins` (page 83)
Copies a range of line origins for a frame.

Drawing the Frame

`CTFrameDraw` (page 82)
Draws an entire frame into a context.

Getting the Type Identifier

`CTFrameGetTypeID` (page 84)
Returns the type identifier for the CTFrame opaque type.

Functions

CTFrameDraw

Draws an entire frame into a context.

```
void CTFrameDraw( CTFrameRef frame, CGContextRef context );
```

Parameters

frame
The frame to draw.

context
The context in which to draw the frame.

Discussion
If both the frame and the context are valid, the frame is drawn in the context. This call can leave the context in any state and does not flush it after the draw operation.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetFrameAttributes

Returns the frame attributes used to create the frame.

```
CFDictionaryRef CTFrameGetFrameAttributes( CTFrameRef frame );
```

Parameters

frame
The frame whose attributes are returned.

Return Value
A reference to a CFDictionary object containing the frame attributes that were used to create the frame, or, if the frame was created without any frame attributes, `NULL`.

Discussion
You can create a frame with an attributes dictionary to control various aspects of the framing process. These attributes are different from the ones used to create an attributed string.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetLineOrigins

Copies a range of line origins for a frame.

```
void CTFrameGetLineOrigins( CTFrameRef frame, CFRange range, CGPoint origins[] );
```

Parameters

frame

The frame whose line origin array is copied.

range

The range of line origins you wish to copy. If the length of the range is 0, then the copy operation continues from the start index of the range to the last line origin.

origins

The buffer to which the origins are copied. The buffer must have at least as many elements as specified by range's length.

Discussion
This function copies a range of `CGPoint` structures. Each `CGPoint` is the origin of the corresponding line in the array of lines returned by `CTFrameGetLines` (page 83), relative to the origin of the frame's path. The maximum number of line origins returned by this function is the count of the array of lines.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetLines

Returns an array of lines stored in the frame.

```
CFArrayRef CTFrameGetLines( CTFrameRef frame );
```

Parameters

frame

The frame whose line array is returned.

Return Value
A CFArray object containing the CTLine objects that make up the frame, or, if there are no lines in the frame, an array with no elements.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetPath

Returns the path used to create the frame.

```
CGPathRef CTFrameGetPath( CTFrameRef frame );
```

Parameters

frame

The frame whose path is returned.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetStringRange

Returns the range of characters originally requested to fill the frame.

```
CFRange CTFrameGetStringRange( CTFrameRef frame );
```

Parameters

frame

The frame whose character range is returned.

Return Value
A `CFRange` structure containing the backing store range of characters that were originally requested to fill the frame, or, if the function call is not successful, an empty range.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetTypeID

Returns the type identifier for the CTFrame opaque type.

```
CFTypeID CTFrameGetTypeID( void );
```

Return Value
The type identifier for the CTFrame opaque type.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFrame.h`

CTFrameGetVisibleStringRange

Returns the range of characters that actually fit in the frame.

```
CFRange CTFrameGetVisibleStringRange( CTFrameRef frame );
```

Parameters

frame

The frame whose visible character range is returned.

Return Value

A `CFRange` structure containing the backing store range of characters that fit into the frame, or if the function call is not successful or no characters fit in the frame, an empty range.

Discussion

This function can be used to cascade frames, because it returns the range of characters that can be seen in the frame. The next frame would start where this frame ends.

Availability

Available in iOS 3.2 and later.

Declared In

`CTFrame.h`

Data Types

CTFrameRef

A reference to a Core text frame object.

```
typedef const struct __CTFrame *CTFrameRef;
```

Availability

Available in iOS 3.2 and later.

Declared In

`CTFrame.h`

Constants

CTFrameProgression

These constants specify frame progression types.

```
enum{
    kCTFrameProgressionTopToBottom = 0,
    kCTFrameProgressionRightToLeft = 1
};
typedef uint32_t CTFrameProgression;
```

Constants

`kCTFrameProgressionTopToBottom`

Lines are stacked top to bottom for horizontal text.

Available in iOS 3.2 and later.

Declared in `CTFrame.h`.

`kCTFrameProgressionRightToLeft`

Lines are stacked right to left for vertical text.

Available in iOS 3.2 and later.

Declared in `CTFrame.h`.

Discussion

The lines of text within a frame may be stacked for either horizontal or vertical text. Values are enumerated for each stacking type supported by CTFrame. Frames created with a progression type specifying vertical text rotate lines 90 degrees counterclockwise when drawing.

Declared In

`CTFrame.h`

kCTFrameProgressionAttributeName

Specifies progression for a frame.

```
const CFStringRef kCTFrameProgressionAttributeName;
```

Constants

`kCTFrameProgressionAttributeName`

A `CFNumberRef` object containing a "CTFrameProgression" (page 85) constant. The default is `kCTFrameProgressionTopToBottom`.

Available in iOS 3.2 and later.

Declared in `CTFrame.h`.

Discussion

This value determines the line-stacking behavior for a frame and does not affect the appearance of the glyphs within that frame.

Declared In

`CTFrame.h`

CTFramesetter Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTFramesetter.h

Overview

The CTFramesetter opaque type is used to generate text frames. That is, CTFramesetter is an object factory for CTFrame objects.

The framesetter takes an attributed string object and a shape descriptor object and calls into the typesetter to create line objects that fill that shape. The output is a frame object containing an array of lines. The frame can then draw itself directly into the current graphic context.

Functions by Task

Creating a Framesetter

`CTFramesetterCreateWithAttributedString` (page 88)
Creates an immutable framesetter object from an attributed string.

Creating Frames

`CTFramesetterCreateFrame` (page 88)
Creates an immutable frame using a framesetter.

`CTFramesetterGetTypesetter` (page 89)
Returns the typesetter object being used by the framesetter.

Frame Sizing

`CTFramesetterSuggestFrameSizeWithConstraints` (page 90)
Determines the frame size needed for a string range.

Getting the Type Identifier

`CTFramesetterGetTypeID` (page 89)
Returns the Core Foundation type identifier of the framesetter object.

Functions

CTFramesetterCreateFrame

Creates an immutable frame using a framesetter.

```
CTFrameRef CTFramesetterCreateFrame( CTFramesetterRef framesetter, CFRange
stringRange, CGPathRef path, CFDictionaryRef frameAttributes );
```

Parameters

framesetter
The framesetter used to create the frame.

stringRange
The range, of the attributed string that was used to create the framesetter, that is to be typeset in lines fitted into the frame. If the length portion of the range is set to `0`, then the framesetter continues to add lines until it runs out of text or space.

path
A CGPath object that specifies the shape of the frame.

frameAttributes
Additional attributes that control the frame filling process can be specified here, or `NULL` if there are no such attributes.

Return Value
A reference to a new CTFrame object if the call was successful; otherwise, `NULL`.

Discussion
This call creates a frame full of glyphs in the shape of the path provided by the *path* parameter. The framesetter continues to fill the frame until it either runs out of text or it finds that text no longer fits.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

CTFramesetterCreateWithAttributedString

Creates an immutable framesetter object from an attributed string.

```
CTFramesetterRef CTFramesetterCreateWithAttributedString( CFAttributedStringRef
string );
```

Parameters

string

The attributed string with which to construct the framesetter object.

Return Value
A reference to a CTFramesetter object if the call was successful; otherwise, `NULL`.

Discussion
The resultant framesetter object can be used to create and fill text frames with the `CTFramesetterCreateFrame` (page 88) call.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

CTFramesetterGetTypeID

Returns the Core Foundation type identifier of the framesetter object.

```
CFTypeID CTFramesetterGetTypeID( void );
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

CTFramesetterGetTypesetter

Returns the typesetter object being used by the framesetter.

```
CTTypesetterRef CTFramesetterGetTypesetter( CTFramesetterRef framesetter );
```

Parameters

framesetter

The framesetter from which a typesetter is requested.

Return Value
A reference to a CTTypesetter object if the call was successful; otherwise, `NULL`. The framesetter maintains a reference to the returned object, which should not be released by the caller.

Discussion
Each framesetter uses a typesetter internally to perform line breaking and other contextual analysis based on the characters in a string; this function returns the typesetter being used by a particular framesetter in case the caller would like to perform other operations on that typesetter.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

CTFramesetterSuggestFrameSizeWithConstraints

Determines the frame size needed for a string range.

```
CGSize CTFramesetterSuggestFrameSizeWithConstraints(
    CTFramesetterRef framesetter,
    CFRange stringRange,
    CFDictionaryRef frameAttributes,
    CGSize constraints,
    CFRange* fitRange
);
```

Parameters

framesetter
The framesetter used for measuring the frame size.

stringRange
The string range to which the frame size applies. The string range is a range over the string used to create the framesetter. If the length portion of the range is set to `0`, then the framesetter continues to add lines until it runs out of text or space.

frameAttributes
Additional attributes that control the frame filling process, or `NULL` if there are no such attributes.

constraints
The width and height to which the frame size is constrained. A value of `CGFLOAT_MAX` for either dimension indicates that it should be treated as unconstrained.

fitRange
On return, contains the range of the string that actually fit in the constrained size.

Return Value
The actual dimensions for the given string range and constraints.

Discussion
This function can be used to determine how much space is needed to display a string, optionally by constraining the space along either dimension.

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

Data Types

CTFramesetterRef

A reference to a Core Foundation framesetter object.

```
typedef const struct __CTFramesetter *CTFramesetterRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTFramesetter.h`

CTGlyphInfo Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTGlyphInfo.h

Overview

The CTGlyphInfo opaque type enables you to override a font's specified mapping from Unicode to the glyph ID.

Functions by Task

Getting the GlyphInfo Type

`CTGlyphInfoGetTypeID` (page 96)
Returns the Core Foundation type identifier of the glyph info object

Creating GlyphInfo Objects

`CTGlyphInfoCreateWithGlyphName` (page 95)
Creates an immutable glyph info object with a glyph name.

`CTGlyphInfoCreateWithGlyph` (page 94)
Creates an immutable glyph info object with a glyph index.

`CTGlyphInfoCreateWithCharacterIdentifier` (page 94)
Creates an immutable glyph info object with a character identifier.

Getting GlyphInfo Data

`CTGlyphInfoGetGlyphName` (page 96)
Gets the glyph name for a glyph info object if that object exists.

`CTGlyphInfoGetCharacterIdentifier` (page 96)
Gets the character identifier for a glyph info object.

`CTGlyphInfoGetCharacterCollection` (page 95)
Gets the character collection for a glyph info object.

Functions

CTGlyphInfoCreateWithCharacterIdentifier

Creates an immutable glyph info object with a character identifier.

```
CTGlyphInfoRef CTGlyphInfoCreateWithCharacterIdentifier( CGFontIndex cid,
CTCharacterCollection collection, CFStringRef baseString );
```

Parameters

cid

A character identifier.

collection

A character collection identifier.

baseString

The part of the string the returned object is intended to override.

Return Value

A valid reference to an immutable CTGlyphInfo object if glyph info creation was successful; otherwise, `NULL`.

Discussion

This function creates an immutable glyph info object for a character identifier and a character collection.

Availability

Available in iOS 3.2 and later.

Declared In

`CTGlyphInfo.h`

CTGlyphInfoCreateWithGlyph

Creates an immutable glyph info object with a glyph index.

```
CTGlyphInfoRef CTGlyphInfoCreateWithGlyph( CGGlyph glyph, CTFontRef font, CFStringRef
 baseString );
```

Parameters

glyph

The index of the glyph.

font

The font to be associated with the returned CTGlyphInfo object.

baseString

The part of the string the returned object is intended to override.

Return Value

A valid reference to an immutable CTGlyphInfo object, If glyph info creation was successful; otherwise, `NULL`.

Discussion

This function creates an immutable glyph info object for a glyph index using a specified font.

Availability

Available in iOS 3.2 and later.

Declared In
`CTGlyphInfo.h`

CTGlyphInfoCreateWithGlyphName

Creates an immutable glyph info object with a glyph name.

```
CTGlyphInfoRef CTGlyphInfoCreateWithGlyphName( CFStringRef glyphName, CTFontRef
font, CFStringRef baseString );
```

Parameters

glyphName
The name of the glyph.

font
The font to be associated with the returned CTGlyphInfo object.

baseString
The part of the string the returned object is intended to override.

Return Value
A valid reference to an immutable CTGlyphInfo object if glyph info creation was successful; otherwise, `NULL`.

Discussion
This function creates an immutable glyph info object for a glyph name such as `copyright` using a specified font.

Availability
Available in iOS 3.2 and later.

Declared In
`CTGlyphInfo.h`

CTGlyphInfoGetCharacterCollection

Gets the character collection for a glyph info object.

```
CTCharacterCollection CTGlyphInfoGetCharacterCollection( CTGlyphInfoRef glyphInfo
 );
```

Parameters

glyphInfo
The glyph info from which to get the character collection. May not be `NULL`.

Return Value
The character collection of the given glyph info object.

Discussion
If the glyph info object was created with a glyph name or a glyph index, its character collection is `kCTIdentityMappingCharacterCollection` (page 97).

Availability
Available in iOS 3.2 and later.

Declared In
`CTGlyphInfo.h`

CTGlyphInfoGetCharacterIdentifier

Gets the character identifier for a glyph info object.

```
CGFontIndex CTGlyphInfoGetCharacterIdentifier( CTGlyphInfoRef glyphInfo );
```

Parameters

glyphInfo

The glyph info from which to get the character identifier. May not be `NULL`.

Return Value

The character identifier of the given glyph info object.

Availability

Available in iOS 3.2 and later.

Declared In

`CTGlyphInfo.h`

CTGlyphInfoGetGlyphName

Gets the glyph name for a glyph info object if that object exists.

```
CFStringRef CTGlyphInfoGetGlyphName( CTGlyphInfoRef glyphInfo );
```

Parameters

glyphInfo

The glyph info from which to get the glyph name. May not be `NULL`.

Return Value

A glyph name if the glyph info object was created; otherwise, `NULL`.

Availability

Available in iOS 3.2 and later.

Declared In

`CTGlyphInfo.h`

CTGlyphInfoGetTypeID

Returns the Core Foundation type identifier of the glyph info object

```
CFTypeID CTGlyphInfoGetTypeID( void );
```

Availability

Available in iOS 3.2 and later.

Declared In

`CTGlyphInfo.h`

Data Types

CTGlyphInfoRef

A reference to a glyph info object.

```
typedef const struct __CTGlyphInfo *CTGlyphInfoRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTGlyphInfo.h`

Constants

CTCharacterCollection

These constants specify character collections.

```
enum{ kCTIdentityMappingCharacterCollection = 0,
kCTAdobeCNS1CharacterCollection = 1,
kCTAdobeGB1CharacterCollection = 2,
kCTAdobeJapan1CharacterCollection = 3,
kCTAdobeJapan2CharacterCollection = 4,
kCTAdobeKorea1CharacterCollection = 5};
typedef uint16_t CTCharacterCollection;
```

Constants

`kCTIdentityMappingCharacterCollection`
The character identifier is equal to the `CGGlyph` glyph index.
Available in iOS 3.2 and later.
Declared in `CTGlyphInfo.h`.

`kCTAdobeCNS1CharacterCollection`
The Adobe-CNS1 mapping.
Available in iOS 3.2 and later.
Declared in `CTGlyphInfo.h`.

`kCTAdobeGB1CharacterCollection`
The Adobe-GB1 mapping.
Available in iOS 3.2 and later.
Declared in `CTGlyphInfo.h`.

`kCTAdobeJapan1CharacterCollection`
The Adobe-Japan1 mapping.
Available in iOS 3.2 and later.
Declared in `CTGlyphInfo.h`.

`kCTAdobeJapan2CharacterCollection`
The Adobe-Japan2 mapping.

Available in iOS 3.2 and later.

Declared in `CTGlyphInfo.h`.

`kCTAdobeKorea1CharacterCollection`
The Adobe-Korea1 mapping.

Available in iOS 3.2 and later.

Declared in `CTGlyphInfo.h`.

Declared In
`CTGlyphInfo.h`

CTLine Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTLine.h

Overview

The CTLine opaque type represents a line of text.

A CTLine object contains an array of glyph runs. Line objects are created by the typesetter during a framesetting operation and can draw themselves directly into a graphics context.

Functions by Task

Creating Lines

`CTLineCreateWithAttributedString` (page 101)
Creates a single immutable line object directly from an attributed string.

`CTLineCreateTruncatedLine` (page 101)
Creates a truncated line from an existing line.

`CTLineCreateJustifiedLine` (page 100)
Creates a justified line from an existing line.

Drawing the Line

`CTLineDraw` (page 102)
Draws a complete line.

Getting Line Data

`CTLineGetGlyphCount` (page 102)
Returns the total glyph count for the line object.

`CTLineGetGlyphRuns` (page 103)
Returns the array of glyph runs that make up the line object.

`CTLineGetStringRange` (page 105)
Gets the range of characters that originally spawned the glyphs in the line.

`CTLineGetPenOffsetForFlush` (page 104)
Gets the pen offset required to draw flush text.

Measuring Lines

`CTLineGetImageBounds` (page 103)
Calculates the image bounds for a line.

`CTLineGetTypographicBounds` (page 106)
Calculates the typographic bounds of a line.

`CTLineGetTrailingWhitespaceWidth` (page 106)
Returns the trailing whitespace width for a line.

Getting Line Positioning

`CTLineGetStringIndexForPosition` (page 105)
Performs hit testing.

`CTLineGetOffsetForStringIndex` (page 104)
Determines the graphical offset or offsets for a string index.

Getting the Type Identifier

`CTLineGetTypeID` (page 106)
Returns the Core Foundation type identifier of the line object.

Functions

CTLineCreateJustifiedLine

Creates a justified line from an existing line.

```
CTLineRef CTLineCreateJustifiedLine( CTLineRef line, CGFloat justificationFactor,
 double justificationWidth );
```

Parameters

line
The line from which to create a justified line.

justificationFactor
Full or partial justification. When set to `1.0` or greater, full justification is performed. If this parameter is set to less than `1.0`, varying degrees of partial justification are performed. If it is set to `0` or less, no justification is performed.

justificationWidth
The width to which the resultant line is justified. If *justificationWidth* is less than the actual width of the line, then negative justification is performed (that is, glyphs are squeezed together).

Return Value
A reference to a justified CTLine object if the call was successful; otherwise, NULL.

Availability
Available in iOS 3.2 and later.

Declared In
CTLine.h

CTLineCreateTruncatedLine

Creates a truncated line from an existing line.

```
CTLineRef CTLineCreateTruncatedLine( CTLineRef line, double width,
CTLineTruncationType truncationType, CTLineRef truncationToken );
```

Parameters

line
The line from which to create a truncated line.

width
The width at which truncation begins. The line is truncated if its width is greater than the width passed in this parameter.

truncationType
The type of truncation to perform if needed. See "CTLineTruncationType" (page 107) for possible values.

truncationToken
This token is added at the point where truncation took place, to indicate that the line was truncated. Usually, the truncation token is the ellipsis character (U+2026). If this parameter is set to NULL, then no truncation token is used and the line is simply cut off.

Return Value
A reference to a truncated CTLine object if the call was successful; otherwise, NULL.

Discussion
The line specified in *truncationToken* should have a width less than the width specified by the *width* parameter. If the width of the line specified in *truncationToken* is greater than *width* and truncation is needed, the function returns NULL.

Availability
Available in iOS 3.2 and later.

Declared In
CTLine.h

CTLineCreateWithAttributedString

Creates a single immutable line object directly from an attributed string.

```
CTLineRef CTLineCreateWithAttributedString( CFAttributedStringRef string );
```

Parameters

string

The string from which the line is created.

Return Value

A reference to a CTLine object if the call was successful; otherwise, `NULL`.

Discussion

This function allows clients who need very simple line generation to create a line without creating a typesetter object. The typesetting is done under the hood. Without a typesetter object, the line cannot be properly broken. However, for simple things like text labels, line breaking is not an issue.

Availability

Available in iOS 3.2 and later.

Declared In

`CTLine.h`

CTLineDraw

Draws a complete line.

```
void CTLineDraw( CTLineRef line, CGContextRef context );
```

Parameters

line

The line to draw.

context

The context into which the line is drawn.

Discussion

This is a convenience function because the line could be drawn run-by-run by getting the glyph runs, getting the glyphs out of them, and calling a function such as `CGContextShowGlyphsAtPositions`. This call can leave the graphics context in any state and does not flush the context after the draw operation.

Availability

Available in iOS 3.2 and later.

Declared In

`CTLine.h`

CTLineGetGlyphCount

Returns the total glyph count for the line object.

```
CFIndex CTLineGetGlyphCount( CTLineRef line );
```

Parameters

line

The line whose glyph count is returned.

Return Value
The total glyph count for the line passed in.

Discussion
The total glyph count is equal to the sum of all of the glyphs in the glyph runs forming the line.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetGlyphRuns

Returns the array of glyph runs that make up the line object.

```
CFArrayRef CTLineGetGlyphRuns( CTLineRef line );
```

Parameters

line
The line whose glyph run array is returned.

Return Value
A `CFArrayRef` containing the CTRun objects that make up the line.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetImageBounds

Calculates the image bounds for a line.

```
CGRect CTLineGetImageBounds( CTLineRef line, CGContextRef context );
```

Parameters

line
The line whose image bounds are calculated.

context
The context for which the image bounds are calculated. This is required because the context could have settings in it that would cause changes in the image bounds.

Return Value
A rectangle that tightly encloses the paths of the line's glyphs, or, if the line or context is invalid, `CGRectNull`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetOffsetForStringIndex

Determines the graphical offset or offsets for a string index.

```
CGFloat CTLineGetOffsetForStringIndex( CTLineRef line, CFIndex charIndex, CGFloat*
 secondaryOffset );
```

Parameters

line

The line from which the offset is requested.

charIndex

The string index corresponding to the desired position.

secondaryOffset

On output, the secondary offset along the baseline for *charIndex*. When a single caret is sufficient for a string index, this value will be the same as the primary offset, which is the return value of this function. May be `NULL`.

Return Value

The primary offset along the baseline for *charIndex*, or `0.0` if the line does not support string access.

Discussion

This function returns the graphical offset or offsets corresponding to a string index, suitable for movement between adjacent lines or for drawing a custom caret. For moving between adjacent lines, the primary offset can be adjusted for any relative indentation of the two lines; a `CGPoint` constructed with the adjusted offset for its `x` value and `0.0` for its `y` value is suitable for passing to `CTLineGetStringIndexForPosition` (page 105). For drawing a custom caret, the returned primary offset corresponds to the portion of the caret that represents the visual insertion location for a character whose direction matches the line's writing direction.

Availability

Available in iOS 3.2 and later.

Declared In

`CTLine.h`

CTLineGetPenOffsetForFlush

Gets the pen offset required to draw flush text.

```
double CTLineGetPenOffsetForFlush( CTLineRef line, CGFloat flushFactor, double
flushWidth );
```

Parameters

line

The line from which to obtain a flush position.

flushFactor

Determines the type of flushness. A *flushFactor* of `0` or less indicates left flush. A *flushFactor* of `1.0` or more indicates right flush. Flush factors between `0` and `1.0` indicate varying degrees of center flush, with a value of `0.5` being totally center flush.

flushWidth

Specifies the width to which the flushness operation should apply.

Return Value

The offset from the current pen position for the flush operation.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetStringIndexForPosition

Performs hit testing.

```
CFIndex CTLineGetStringIndexForPosition( CTLineRef line, CGPoint position );
```

Parameters

`line`
The line being examined.

`position`
The location of the mouse click relative to the line's origin.

Return Value
The string index for the position, or if the line does not support string access, `kCFNotFound`. Relative to the line's string range, this value can be no less than the first string index and no greater than the last string index plus 1.

Discussion
This function can be used to determine the string index for a mouse click or other event. This string index corresponds to the character before which the next character should be inserted. This determination is made by analyzing the string from which a typesetter was created and the corresponding glyphs as embodied by a particular line.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetStringRange

Gets the range of characters that originally spawned the glyphs in the line.

```
CFRange CTLineGetStringRange( CTLineRef line );
```

Parameters

`line`
The line from which to obtain the string range.

Return Value
A `CFRange` structure that contains the range over the backing store string that spawned the glyphs, or if the function fails for any reason, an empty range.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

CTLineGetTrailingWhitespaceWidth

Returns the trailing whitespace width for a line.

```
double CTLineGetTrailingWhitespaceWidth( CTLineRef line );
```

Parameters

line

The line whose trailing whitespace width is calculated.

Return Value

The width of the line's trailing whitespace. If the line is invalid, this function will always return zero.

Discussion

Creating a line for a width can result in a line that is actually longer than the desired width due to trailing whitespace. Although this is typically not an issue due to whitespace being invisible, this function can be used to determine what amount of a line's width is due to trailing whitespace.

Availability

Available in iOS 3.2 and later.

Declared In

`CTLine.h`

CTLineGetTypeID

Returns the Core Foundation type identifier of the line object.

```
CFTypeID CTLineGetTypeID( void );
```

Availability

Available in iOS 3.2 and later.

Declared In

`CTLine.h`

CTLineGetTypographicBounds

Calculates the typographic bounds of a line.

```
double CTLineGetTypographicBounds( CTLineRef line, CGFloat* ascent, CGFloat* descent,
 CGFloat* leading );
```

Parameters

line

The line whose typographic bounds are calculated.

ascent

On output, the ascent of the line. This parameter can be set to `NULL` if not needed.

descent

On output, the descent of the line. This parameter can be set to `NULL` if not needed.

leading

On output, the leading of the line. This parameter can be set to `NULL` if not needed.

Return Value
The typographic width of the line. If the line is invalid, this function returns `0`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

Data Types

CTLineRef

A reference to a line object.

```
typedef const struct __CTLine *CTLineRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTLine.h`

Constants

CTLineTruncationType

Truncation types required by the `CTLineCreateTruncatedLine` (page 101) function to tell the truncation engine which type of truncation is being requested.

```
enum{
    kCTLineTruncationStart = 0,
    kCTLineTruncationEnd = 1,
    kCTLineTruncationMiddle = 2
};
typedef uint32_t CTLineTruncationType;
```

Constants

`kCTLineTruncationStart`
Truncate the beginning of the line, leaving the end portion visible.
Available in iOS 3.2 and later.
Declared in `CTLine.h`.

`kCTLineTruncationEnd`
Truncate the end of the line, leaving the start portion visible.
Available in iOS 3.2 and later.
Declared in `CTLine.h`.

`kCTLineTruncationMiddle`

Truncate the middle of the line, leaving both the start and the end portions visible.

Available in iOS 3.2 and later.

Declared in `CTLine.h`.

Declared In

`CTLine.h`

CTParagraphStyle Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTParagraphStyle.h

Overview

The CTParagraphStyle opaque type represents paragraph or ruler attributes in an attributed string.

A paragraph style object represents a complex attribute value in an attributed string, storing a number of subattributes that affect paragraph layout for the characters of the string. Among these subattributes are alignment, tab stops, writing direction, line-breaking mode, and indentation settings.

Functions by Task

Creating Paragraph Styles

`CTParagraphStyleCreate` (page 110)
Creates an immutable paragraph style.

`CTParagraphStyleCreateCopy` (page 110)
Creates an immutable copy of a paragraph style.

Getting the Value of a Style Specifier

`CTParagraphStyleGetValueForSpecifier` (page 111)
Obtains the current value for a single setting specifier.

Getting the Type Identifier

`CTParagraphStyleGetTypeID` (page 111)
Returns the Core Foundation type identifier of the paragraph style object.

Functions

CTParagraphStyleCreate

Creates an immutable paragraph style.

```
CTParagraphStyleRef CTParagraphStyleCreate( const CTParagraphStyleSetting* settings,
 CFIndex settingCount );
```

Parameters

settings

The settings with which to preload the paragraph style. If you want to specify the default set of settings, set this parameter to `NULL`.

settingCount

The number of settings that you have specified in the *settings* parameter. This must be greater than or equal to `0`.

Return Value

A valid reference to an immutable CTParagraphStyle object, If the paragraph style creation was successful; otherwise, `NULL`.

Discussion

Using this function is the easiest and most efficient way to create a paragraph style. Paragraph styles should be kept immutable for totally lock-free operation. If an invalid paragraph style setting specifier is passed into the *settings* parameter, nothing bad will happen, but you will be unable to query for this value. The reason is to allow backward compatibility with style setting specifiers that may be introduced in future versions.

Availability

Available in iOS 3.2 and later.

Declared In

`CTParagraphStyle.h`

CTParagraphStyleCreateCopy

Creates an immutable copy of a paragraph style.

```
CTParagraphStyleRef CTParagraphStyleCreateCopy( CTParagraphStyleRef paragraphStyle
 );
```

Parameters

paragraphStyle

The style to copy. This parameter may not be `NULL`.

Return Value

A valid reference to an immutable CTParagraphStyle object that is a copy of the one passed into *paragraphStyle*, If the *paragraphStyle* reference is valid; otherwise `NULL`, if any error occurred, including being supplied with an invalid reference.

Availability

Available in iOS 3.2 and later.

Declared In
`CTParagraphStyle.h`

CTParagraphStyleGetTypeID

Returns the Core Foundation type identifier of the paragraph style object.

```
CFTypeID CTParagraphStyleGetTypeID( void );
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTParagraphStyle.h`

CTParagraphStyleGetValueForSpecifier

Obtains the current value for a single setting specifier.

```
bool CTParagraphStyleGetValueForSpecifier( CTParagraphStyleRef paragraphStyle,
CTParagraphStyleSpecifier spec, size_t valueBufferSize, void* valueBuffer );
```

Parameters

paragraphStyle
The paragraph style from which to get the value. This parameter may not be `NULL`.

spec
The setting specifier for which to get the value.

valueBufferSize
The size of the buffer pointed to by the *valueBuffer* parameter. This value must be at least as large as the size the required by the `CTParagraphStyleSpecifier` (page 115) value set in the *spec* parameter.

valueBuffer
On output, the requested setting value. The buffer's size needs to be at least as large as the value passed into *valueBufferSize*. This parameter is required and may not be `NULL`.

Return Value
`True` if *valueBuffer* was successfully filled; otherwise, `False`, indicating that one or more of the parameters are not valid.

Discussion
This function returns the current value of the specifier whether or not the user actually set it. If the user did not set the specifier, this function returns the default value. If an invalid paragraph style setting specifier is passed into the *spec* parameter, nothing bad happens, and the buffer value is simply zeroed out. The reason is to allow backward compatibility with style setting specifiers that may be introduced in future versions.

Availability
Available in iOS 3.2 and later.

Declared In
`CTParagraphStyle.h`

Data Types

CTParagraphStyleSetting

This structure is used to alter the paragraph style.

```
typedef struct CTParagraphStyleSetting{ CTParagraphStyleSpecifier spec; size_t
valueSize; const void* value;} CTParagraphStyleSetting;
```

Fields

`spec`

The specifier of the setting. See "CTParagraphStyleSpecifier" (page 115) for possible values.

`valueSize`

The size of the value pointed to by the *`value`* field. This value must match the size of the value required by the `CTParagraphStyleSpecifier` set in the *`spec`* field.

`value`

A reference to the value of the setting specified by the *`spec`* field. The value must be in the proper range for the *`spec`* value and at least as large as the size specified in *`valueSize`*.

Availability
Available in iOS 3.2 and later.

Declared In
`CTParagraphStyle.h`

CTParagraphStyleRef

A reference to a Core Text paragraph style.

```
typedef const struct __CTParagraphStyle *CTParagraphStyleRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTParagraphStyle.h`

Constants

CTTextAlignment

These constants specify text alignment.

```
enum{
kCTLeftTextAlignment = 0,
kCTRightTextAlignment = 1,
kCTCenterTextAlignment = 2,
kCTJustifiedTextAlignment = 3,
kCTNaturalTextAlignment = 4
};
typedef uint8_t CTTextAlignment;
```

Constants

`kCTLeftTextAlignment`
Text is visually left aligned.
Available in iOS 3.2 and later.
Declared in `CTParagraphStyle.h`.

`kCTRightTextAlignment`
Text is visually right aligned.
Available in iOS 3.2 and later.
Declared in `CTParagraphStyle.h`.

`kCTCenterTextAlignment`
Text is visually center aligned.
Available in iOS 3.2 and later.
Declared in `CTParagraphStyle.h`.

`kCTJustifiedTextAlignment`
Text is fully justified. The last line in a paragraph is naturally aligned.
Available in iOS 3.2 and later.
Declared in `CTParagraphStyle.h`.

`kCTNaturalTextAlignment`
Text uses the natural alignment of the text's script.
Available in iOS 3.2 and later.
Declared in `CTParagraphStyle.h`.

Declared In
`CTParagraphStyle.h`

CTLineBreakMode

These constants specify what happens when a line is too long for its frame.

```
enum{
kCTLineBreakByWordWrapping = 0,
kCTLineBreakByCharWrapping = 1,
kCTLineBreakByClipping = 2,
kCTLineBreakByTruncatingHead = 3,
kCTLineBreakByTruncatingTail = 4,
kCTLineBreakByTruncatingMiddle = 5
};
typedef uint8_t CTLineBreakMode;
```

Constants

`kCTLineBreakByWordWrapping`
Wrapping occurs at word boundaries unless the word itself doesn't fit on a single line.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByCharWrapping`
Wrapping occurs before the first character that doesn't fit.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByClipping`
Lines are simply not drawn past the edge of the frame.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByTruncatingHead`
Each line is displayed so that the end fits in the frame and the missing text is indicated by an ellipsis glyph.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByTruncatingTail`
Each line is displayed so that the beginning fits in the container and the missing text is indicated by an ellipsis glyph.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByTruncatingMiddle`
Each line is displayed so that the beginning and end fit in the container and the missing text is indicated by an ellipsis glyph in the middle.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

Declared In
`CTParagraphStyle.h`

CTWritingDirection

These constants specify the writing direction.

```
enum{
kCTWritingDirectionNatural = -1,
kCTWritingDirectionLeftToRight = 0,
kCTWritingDirectionRightToLeft = 1
};
typedef int8_t CTWritingDirection;
```

Constants

`kCTWritingDirectionNatural`
The writing direction is algorithmically determined using the Unicode Bidirectional Algorithm rules P2 and P3.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTWritingDirectionLeftToRight`
The writing direction is left to right.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTWritingDirectionRightToLeft`
The writing direction is right to left.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

Declared In
`CTParagraphStyle.h`

CTParagraphStyleSpecifier

These constants are used to query and modify the CTParagraphStyle object.

```
enum{
kCTParagraphStyleSpecifierAlignment = 0,
kCTParagraphStyleSpecifierFirstLineHeadIndent = 1,
kCTParagraphStyleSpecifierHeadIndent = 2,
kCTParagraphStyleSpecifierTailIndent = 3,
kCTParagraphStyleSpecifierTabStops = 4,
kCTParagraphStyleSpecifierDefaultTabInterval = 5,
kCTParagraphStyleSpecifierLineBreakMode = 6,
kCTParagraphStyleSpecifierLineHeightMultiple = 7,
kCTParagraphStyleSpecifierMaximumLineHeight = 8,
kCTParagraphStyleSpecifierMinimumLineHeight = 9,
kCTParagraphStyleSpecifierLineSpacing = 10,
kCTParagraphStyleSpecifierParagraphSpacing = 11,
kCTParagraphStyleSpecifierParagraphSpacingBefore = 12,
kCTParagraphStyleSpecifierBaseWritingDirection = 13,
kCTParagraphStyleSpecifierCount = 14
};
typedef uint32_t CTParagraphStyleSpecifier;
```

Constants

`kCTParagraphStyleSpecifierAlignment`

The text alignment. Natural text alignment is realized as left or right alignment, depending on the line sweep direction of the first script contained in the paragraph. Type: `CTTextAlignment` (page 112). Default: `kCTNaturalTextAlignment` (page 113). Application: CTFramesetter.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierFirstLineHeadIndent`

The distance, in points, from the leading margin of a frame to the beginning of the paragraph's first line. This value is always nonnegative. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierHeadIndent`

The distance, in points, from the leading margin of a text container to the beginning of lines other than the first. This value is always nonnegative. Type: `CGFloat` Default: `0.0` Application: CTFramesetter

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierTailIndent`

The distance, in points, from the margin of a frame to the end of lines. If positive, this value is the distance from the leading margin (for example, the left margin in left-to-right text). If `0` or negative, it's the distance from the trailing margin. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierTabStops`

The CTTextTab objects, sorted by location, that define the tab stops for the paragraph style. Type: CFArray of `CTTextTabRef` (page 141). Default: 12 left-aligned tabs, spaced by 28.0 points. Application: CTFramesetter, CTTypesetter.

Available in iOS 3.2 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierDefaultTabInterval`
: The documentwide default tab interval. Tabs after the last specified by `kCTParagraphStyleSpecifierTabStops` are placed at integer multiples of this distance (if positive). Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter, CTTypesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierLineBreakMode`
: The mode that should be used to break lines when laying out the paragraph's text. Type: `CTLineBreakMode` (page 113). Default: `kCTLineBreakByWordWrapping` (page 114). Application: CTFramesetter

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierLineHeightMultiple`
: The line height multiple. The natural line height of the receiver is multiplied by this factor (if positive) before being constrained by minimum and maximum line height. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierMaximumLineHeight`
: The maximum height that any line in the frame will occupy, regardless of the font size or size of any attached graphic. Glyphs and graphics exceeding this height will overlap neighboring lines. A maximum height of `0` implies no line height limit. This value is always nonnegative. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierMinimumLineHeight`
: The minimum height that any line in the frame will occupy, regardless of the font size or size of any attached graphic. This value is always nonnegative. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierLineSpacing`
: The space in points added between lines within the paragraph (commonly known as leading). This value is always nonnegative. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierParagraphSpacing`
: The space added at the end of the paragraph to separate it from the following paragraph. This value is always nonnegative and is determined by adding the previous paragraph's `kCTParagraphStyleSpecifierParagraphSpacing` setting and the current paragraph's `kCTParagraphStyleSpecifierParagraphSpacingBefore` setting. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierParagraphSpacingBefore`
: The distance between the paragraph's top and the beginning of its text content. Type: `CGFloat`. Default: `0.0`. Application: CTFramesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierBaseWritingDirection`
: The base writing direction of the lines. Type: `CTWritingDirection` (page 114). Default: `kCTWritingDirectionNatural` (page 115). Application: CTFramesetter, CTTypesetter.

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierCount`
: The number of style specifiers. The purpose is to simplify validation of style specifiers

  Available in iOS 3.2 and later.

  Declared in `CTParagraphStyle.h`.

Discussion
Each specifier has a type and a default value associated with it. The type must always be observed when setting or fetching the value from the CTParagraphStyle object. In addition, some specifiers affect the behavior of both the framesetter and the typesetter, and others affect the behavior of only the framesetter, as noted in the constant descriptions.

Declared In
`CTParagraphStyle.h`

CTRun Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTRun.h

Overview

The CTRun opaque type represents a glyph run, which is a set of consecutive glyphs sharing the same attributes and direction.

The typesetter creates glyph runs as it produces lines from character strings, attributes, and font objects. That is, a line is constructed of one or more glyphs runs. Glyph runs can draw themselves into a graphic context, if desired, although most users have no need to interact directly with glyph runs.

Functions by Task

Getting Glyph Run Data

`CTRunGetGlyphCount` (page 122)
Gets the glyph count for the run.

`CTRunGetAttributes` (page 122)
Returns the attribute dictionary that was used to create the glyph run.

`CTRunGetStatus` (page 125)
Returns the run's status.

`CTRunGetGlyphsPtr` (page 123)
Returns a direct pointer for the glyph array stored in the run.

`CTRunGetGlyphs` (page 123)
Copies a range of glyphs into a user-provided buffer.

`CTRunGetPositionsPtr` (page 125)
Returns a direct pointer for the glyph position array stored in the run.

`CTRunGetPositions` (page 124)
Copies a range of glyph positions into a user-provided buffer.

`CTRunGetAdvancesPtr` (page 121)
Returns a direct pointer for the glyph advance array stored in the run.

`CTRunGetAdvances` (page 121)
Copies a range of glyph advances into a user-provided buffer.

`CTRunGetStringIndicesPtr` (page 127)
Returns a direct pointer for the string indices stored in the run.

`CTRunGetStringIndices` (page 126)
Copies a range of string indices into a user-provided buffer.

`CTRunGetStringRange` (page 127)
Gets the range of characters that originally spawned the glyphs in the run.

Measuring the Glyph Run

`CTRunGetTypographicBounds` (page 128)
Gets the typographic bounds of the run.

`CTRunGetImageBounds` (page 124)
Calculates the image bounds for a glyph range.

Drawing the Glyph Run

`CTRunDraw` (page 120)
Draws a complete run or part of one.

`CTRunGetTextMatrix` (page 127)
Returns the text matrix needed to draw this run.

Getting the Type Identifier

`CTRunGetTypeID` (page 128)
Returns the Core Foundation type identifier of the run object.

Functions

CTRunDraw

Draws a complete run or part of one.

```
void CTRunDraw (
   CTRunRef run,
   CGContextRef context,
   CFRange range
);
```

Parameters

run
The run to draw.

context
The context into which to draw the run.

range
The portion of the run to draw. If the length of the range is set to `0`, then the draw operation continues from the start index of the range to the end of the run.

Discussion
This is a convenience call, because the run could be drawn by accessing the glyphs. This call can leave the graphics context in any state and does not flush the context after the draw operation.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetAdvances

Copies a range of glyph advances into a user-provided buffer.

```
void CTRunGetAdvances(
    CTRunRef run,
    CFRange range,
    CGSize buffer[]
);
```

Parameters

run
The run whose advances you wish to copy.

range
The range of glyph advances you wish to copy. If the length of the range is set to `0`, then the copy operation continues from the range's start index to the end of the run.

buffer
The buffer to which the glyph advances are copied. The buffer must be allocated to at least the value specified by the range's length.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetAdvancesPtr

Returns a direct pointer for the glyph advance array stored in the run.

```
const CGSize* CTRunGetAdvancesPtr(
    CTRunRef run
);
```

Parameters

run
The run whose advances you wish to access.

Return Value
A valid pointer to an array of `CGSize` structures representing the glyph advance array or `NULL`.

Discussion
The advance array will have a length equal to the value returned by `CTRunGetGlyphCount` (page 122). The caller should be prepared for this function to return `NULL` even if there are glyphs in the stream. Should this function return `NULL`, the caller needs allocate its own buffer and call `CTRunGetAdvances` (page 121) to fetch the advances. Note that advances alone are not sufficient for correctly positioning glyphs in a line, as a run may have a non-identity matrix or the initial glyph in a line may have a non-zero origin; callers should consider using positions instead.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetAttributes

Returns the attribute dictionary that was used to create the glyph run.

```
CFDictionaryRef CTRunGetAttributes (
   CTRunRef run
);
```

Parameters

run
The run for which to return attributes.

Return Value
A valid `CFDictionaryRef` or `NULL` on error or if the run has no attributes.

Discussion
The dictionary returned is either the same one that was set as an attribute dictionary on the original attributed string or a dictionary that has been manufactured by the layout engine. Attribute dictionaries can be manufactured in the case of font substitution or if the run is missing critical attributes.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetGlyphCount

Gets the glyph count for the run.

```
CFIndex CTRunGetGlyphCount (
   CTRunRef run
);
```

Parameters

run
The run for which to return the glyph count.

Return Value
The number of glyphs that the run contains, or if there are no glyphs in this run, a value of `0`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetGlyphs

Copies a range of glyphs into a user-provided buffer.

```
void CTRunGetGlyphs (
   CTRunRef run,
   CFRange range,
   CGGlyph buffer[]
);
```

Parameters

run
The run from which to copy glyphs.

range
The range of glyphs to copy. If the length of the range is set to `0`, then the copy operation continues from the range's start index to the end of the run.

buffer
The buffer the glyphs are copied to. The buffer must be allocated to at least the value specified by the range's length.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetGlyphsPtr

Returns a direct pointer for the glyph array stored in the run.

```
const CGGlyph * CTRunGetGlyphsPtr (
   CTRunRef run
);
```

Parameters

run
The run from which to return glyphs.

Return Value
A valid pointer to an array of `CGGlyph` structures, or `NULL`.

Discussion
The glyph array will have a length equal to the value returned by `CTRunGetGlyphCount` (page 122). The caller should be prepared for this function to return `NULL` even if there are glyphs in the stream. If this function returns `NULL`, the caller must allocate its own buffer and call `CTRunGetGlyphs` to fetch the glyphs.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetImageBounds

Calculates the image bounds for a glyph range.

```
CGRect CTRunGetImageBounds (
   CTRunRef run,
   CGContextRef context,
   CFRange range
);
```

Parameters

run
The run for which to calculate the image bounds.

context
The context for the image bounds being calculated. This is required because the context could have settings in it that would cause changes in the image bounds.

range
The portion of the run to measure. If the length of the range is set to `0`, then the measure operation continues from the start index of the range to the end of the run.

Return Value
A rectangle that tightly encloses the paths of the run's glyphs, or, if *run*, *context*, or *range* is invalid, `CGRectNull`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetPositions

Copies a range of glyph positions into a user-provided buffer.

```
void CTRunGetPositions (
   CTRunRef run,
   CFRange range,
   CGPoint buffer[]
);
```

Parameters

run

The run from which to copy glyph positions.

range

The range of glyph positions to copy. If the length of the range is set to `0`, then the copy operation will continue from the start index of the range to the end of the run.

buffer

The buffer to which the glyph positions are copied. The buffer must be allocated to at least the value specified by the range's length.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetPositionsPtr

Returns a direct pointer for the glyph position array stored in the run.

```
const CGPoint * CTRunGetPositionsPtr (
   CTRunRef run
);
```

Parameters

run

The run from which to access glyph positions.

Return Value
A valid pointer to an array of `CGPoint` structures, or `NULL`.

Discussion
The glyph positions in a run are relative to the origin of the line containing the run. The position array will have a length equal to the value returned by `CTRunGetGlyphCount` (page 122). The caller should be prepared for this function to return `NULL` even if there are glyphs in the stream. If this function returns `NULL`, the caller must allocate its own buffer and call `CTRunGetPositions` (page 124) to fetch the glyph positions.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetStatus

Returns the run's status.

```
CTRunStatus CTRunGetStatus (
   CTRunRef run
);
```

Parameters

run

The run for which to return the status.

Return Value
The run's status.

Discussion
Runs have status that can be used to expedite certain operations. Knowing the direction and ordering of a run's glyphs can aid in string index analysis, whereas knowing whether the positions reference the identity text matrix can avoid expensive comparisons. This status is provided as a convenience, because this information is not strictly necessary but can be helpful in some circumstances.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetStringIndices

Copies a range of string indices into a user-provided buffer.

```
void CTRunGetStringIndices (
   CTRunRef run,
   CFRange range,
   CFIndex buffer[]
);
```

Parameters

run

The run from which to copy the string indices.

range

The range of string indices to copy. If the length of the range is set to `0`, then the copy operation continues from the range's start index to the end of the run.

buffer

The buffer to which the string indices are copied. The buffer must be allocated to at least the value specified by the range's length.

Discussion
The indices are the character indices that originally spawned the glyphs that make up the run. They can be used to map the glyphs in the run back to the characters in the backing store.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetStringIndicesPtr

Returns a direct pointer for the string indices stored in the run.

```
const CFIndex * CTRunGetStringIndicesPtr (
   CTRunRef run
);
```

Parameters

run

The run for which to return string indices.

Return Value

A valid pointer to an array of `CFIndex` structures, or `NULL`.

Discussion

The indices are the character indices that originally spawned the glyphs that make up the run. They can be used to map the glyphs in the run back to the characters in the backing store. The string indices array will have a length equal to the value returned by `CTRunGetGlyphCount` (page 122). The caller should be prepared for this function to return `NULL` even if there are glyphs in the stream. If this function returns `NULL`, the caller must allocate its own buffer and call `CTRunGetStringIndices` (page 126) to fetch the indices.

Availability

Available in iOS 3.2 and later.

Declared In

`CTRun.h`

CTRunGetStringRange

Gets the range of characters that originally spawned the glyphs in the run.

```
CFRange CTRunGetStringRange (
   CTRunRef run
);
```

Parameters

run

The run for which to access the string range.

Return Value

The range of characters that originally spawned the glyphs, of if *run* is invalid, an empty range.

Availability

Available in iOS 3.2 and later.

Declared In

`CTRun.h`

CTRunGetTextMatrix

Returns the text matrix needed to draw this run.

```
CGAffineTransform CTRunGetTextMatrix (
   CTRunRef run
);
```

Parameters

run

The run object from which to get the text matrix.

Return Value
A `CGAffineTransform` structure.

Discussion
To properly draw the glyphs in a run, the fields `tx` and `ty` of the `CGAffineTransform` returned by this function should be set to the current text position.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetTypeID

Returns the Core Foundation type identifier of the run object.

```
CFTypeID CTRunGetTypeID (
   void
);
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

CTRunGetTypographicBounds

Gets the typographic bounds of the run.

```
double CTRunGetTypographicBounds (
   CTRunRef run,
   CFRange range,
   CGFloat *ascent,
   CGFloat *descent,
   CGFloat *leading
);
```

Parameters

run

The run for which to calculate the typographic bounds.

range

The portion of the run to measure. If the length of the range is set to `0`, then the measure operation continues from the range's start index to the end of the run.

ascent
On output, the ascent of the run. This can be set to `NULL` if not needed.

descent
On output, the descent of the run. This can be set to `NULL` if not needed.

leading
On output, the leading of the run. This can be set to `NULL` if not needed.

Return Value
The typographic width of the run, or if *run* or *range* is invalid, `0`.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

Data Types

CTRunRef

A reference to a run object.

```
typedef const struct __CTRun *CTRunRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTRun.h`

Constants

CTRunStatus

A bitfield passed back by the `CTRunGetStatus` (page 125) function that is used to indicate the disposition of the run.

```
enum{
kCTRunStatusNoStatus = 0,
kCTRunStatusRightToLeft = (1 << 0),
kCTRunStatusNonMonotonic = (1 << 1),
kCTRunStatusHasNonIdentityMatrix = (1 << 2)
};
typedef uint32_t CTRunStatus;
```

Constants

`kCTRunStatusNoStatus`

The run has no special attributes.

Available in iOS 3.2 and later.

Declared in `CTRun.h`.

`kCTRunStatusRightToLeft`

The run proceeds from right to left.

Available in iOS 3.2 and later.

Declared in `CTRun.h`.

`kCTRunStatusNonMonotonic`

The run has been reordered in some way such that the string indices associated with the glyphs are no longer strictly increasing (for left-to-right runs) or decreasing (for right-to-left runs).

Available in iOS 3.2 and later.

Declared in `CTRun.h`.

`kCTRunStatusHasNonIdentityMatrix`

The run requires a specific text matrix to be set in the current Core Graphics context for proper drawing.

Available in iOS 3.2 and later.

Declared in `CTRun.h`.

Declared In

`CTRun.h`

CTRunDelegate Reference

Derived From:	CFType
Framework:	CoreText <<need to check this>>
Declared in	CTRunDelegate.h

Overview

The CTRunDelegate opaque type represents a run delegate, which is assigned to a run (attribute range) to control typographic traits such glyph ascent, glyph descent, and glyph width.

The callbacks defined for CTRunDelegate objects are provided by the owner of a run delegate and are used to modify glyph metrics during layout. The values returned by the delegate are applied to each glyph in the run or runs corresponding to the attribute containing that delegate.

Functions by Task

Creating a Run Delegate

`CTRunDelegateCreate` (page 131)
Creates an immutable instance of a run delegate.

Getting Information About a Run Delegate

`CTRunDelegateGetRefCon` (page 132)
Returns a run delegate's "refCon" value.

`CTRunDelegateGetTypeID` (page 132)
Returns the type of CTRunDelegate objects.

Functions

CTRunDelegateCreate

Creates an immutable instance of a run delegate.

```
CTRunDelegateRef CTRunDelegateCreate(const CTRunDelegateCallbacks* callbacks, void*
 refCon )
```

Parameters

callbacks

A structure holding pointers to the callbacks for this run delegate.

refCon

A constant value associated with the run delegate to identify it.

Return Value
If successful, a reference to an immutable CTRunDelegate object. Otherwise, returns `NULL`.

Discussion
The run-delegate object can be used for reserving space in a line or for eliding the glyphs for a range of text altogether.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

CTRunDelegateGetRefCon

Returns a run delegate's "refCon" value.

```
void* CTRunDelegateGetRefCon(CTRunDelegateRef runDelegate);
```

Parameters

runDelegate

The run delegate object being queried.

Return Value
A constant value associated with the run delegate as an identifier.

Discussion
The run delegate object was created with the returned "refCon" value.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

CTRunDelegateGetTypeID

Returns the type of CTRunDelegate objects.

```
CFTypeID CTRunDelegateGetTypeID( void );
```

Discussion
The return type is a Core Foundation type (CTType).

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

Callbacks by Task

Determining Typographic Traits

`CTRunDelegateGetAscentCallback` (page 134)
Defines a pointer to a function that determines typographic ascent of glyphs in the run.

`CTRunDelegateGetDescentCallback` (page 134)
Defines a pointer to a function that determines typographic descent of glyphs in the run.

`CTRunDelegateGetWidthCallback` (page 134)
Defines a pointer to a function that determines the typographic width of glyphs in the run.

Deallocating the Run Delegate

`CTRunDelegateDeallocateCallback` (page 133)
Defines a pointer to a function that is invoked when a CTRunDelegate object is deallocated.

Callbacks

CTRunDelegateDeallocateCallback

Defines a pointer to a function that is invoked when a CTRunDelegate object is deallocated.

```
typedef void (*CTRunDelegateDeallocateCallback) ( void* refCon );
```

You would declare the deallocation function like this if you were to name it `MyDeallocationCallback`:

```
void MyDeallocationCallback( void* refCon );
```

Parameters
refCon
The reference-constant value supplied to the `CTRunDelegateCreate` (page 131) function when the run delegate was created.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

CTRunDelegateGetAscentCallback

Defines a pointer to a function that determines typographic ascent of glyphs in the run.

```
typedef CGFloat (*CTRunDelegateGetAscentCallback) ( void* refCon );
```

You would declare the get-ascent function like this if you were to name it `MyGetAscentCallback`:

```
CGFloat MyGetAscentCallback( void *refCon );
```

Parameters

refCon

The reference-constant value supplied to the `CTRunDelegateCreate` (page 131) function when the run delegate was created.

Return Value

The typographic ascent of glyphs in the run associated with the run delegate.

Availability

Available in iOS 3.2 and later.

Declared In

`CTRunDelegate.h`

CTRunDelegateGetDescentCallback

Defines a pointer to a function that determines typographic descent of glyphs in the run.

```
typedef CGFloat (*CTRunDelegateGetDescentCallback) ( void* refCon );
```

You would declare the get-ascent function like this if you were to name it `MyGetDescentCallback`:

```
CGFloat MyGetDescentCallback( void *refCon );
```

Parameters

refCon

The reference-constant value supplied to the `CTRunDelegateCreate` (page 131) function when the run delegate was created.

Return Value

The typographic descent of glyphs in the run associated with the run delegate.

Availability

Available in iOS 3.2 and later.

Declared In

`CTRunDelegate.h`

CTRunDelegateGetWidthCallback

Defines a pointer to a function that determines the typographic width of glyphs in the run.

```
typedef CGFloat (*CTRunDelegateGetWidthCallback) ( void* refCon );
```

You would declare the get-width function like this if you were to name it `MyGetWidthCallback`:

```
CGFloat MyGetWidthCallback( void* refCon );
```

Parameters

refCon

The reference-constant value supplied to the `CTRunDelegateCreate` (page 131) function when the run delegate was created.

Return Value

The typographic width of glyphs in the run associated with the run delegate. A value of 0.0 indicates that the glyphs should not be drawn.

Availability

Available in iOS 3.2 and later.

Declared In

`CTRunDelegate.h`

Data Types

CTRunDelegateCallbacks

A structure holding pointers to callbacks implemented by the run delegate.

```
typedef struct
{
    CFIndex                             version;
    CTRunDelegateDeallocateCallback     dealloc;
    CTRunDelegateGetAscentCallback     getAscent;
    CTRunDelegateGetDescentCallback     getDescent;
    CTRunDelegateGetWidthCallback     getWidth;
} CTRunDelegateCallbacks;
```

Fields

`version`

The version number of the callbacks being passed in as a parameter to `CTRunDelegateCreate` (page 131). The initial version is `kCTRunDelegateVersion0` (page 137).

`dealloc`

The callback invoked when the retain count of a CTRunDelegate reaches 0 and the CTRunDelegate is deallocated. This callback may be `NULL`.

`getAscent`

The callback invoked to request the run delegate to determine and return the typographic ascent of glyphs in the run. This callback may be `NULL`, which is equivalent to a *`getAscent`* callback that always returns 0.

`getDescent`

The callback invoked to request the run delegate to determine and return the typographic descent of glyphs in the run. This callback may be `NULL`, which is equivalent to a *`getDescent`* callback that always returns 0.

`getWidth`

The callback invoked to request the run delegate to determine and return the typographic width of glyphs in the run. This callback may be `NULL`, which is equivalent to a *`getWidth`* callback that always returns 0.

Discussion
You pass in a pointer to this structure when you create a CTRunDelegate object with the `CTRunDelegateCreate` (page 131) function. The callbacks defined in this structure are provided by the owner of a run delegate and are used to modify glyph metrics during layout. The values returned by the delegate are applied to each glyph in the run or runs corresponding to the attribute containing that delegate.

See "Callbacks" (page 133) for a discussion of the function-pointer types associated with these callbacks.

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

CTRunDelegateRef

The type of the CTRunDelegate opaque object.

```
typedef const struct __CTRunDelegate * CTRunDelegateRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTRunDelegate.h`

Constants

Run Delegate Versions

The version of the run delegate.

```
enum {
    kCTRunDelegateVersion1 = 1,
    kCTRunDelegateCurrentVersion = kCTRunDelegateVersion0
};
```

Constants

`kCTRunDelegateVersion1`

Version 1 of the run delegate.

Available in iOS 3.2 and later.

Declared in `CTRunDelegate.h`.

`kCTRunDelegateCurrentVersion`

The current version of the run delegate.

Available in iOS 3.2 and later.

Declared in `CTRunDelegate.h`.

Discussion

Set the version field of the `CTRunDelegateCallbacks` (page 135) structure to `kCTRunDelegateCurrentVersion` when creating a CTRunDelegate object with a call to `CTRunDelegateCreate` (page 131).

Declared In

`CTRunDelegate.h`

CTTextTab Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTTextTab.h

Overview

The CTTextTab opaque type represents a tab in a paragraph style, storing an alignment type and location.

Core Text supports four alignment types: left, center, right, and decimal. These alignment types are absolute, not based on the line sweep direction of text. For example, tabbed text is always positioned to the left of a right-aligned tab, whether the line sweep direction is left to right or right to left. A tab's location, on the other hand, is relative to the back margin. A tab set at 1.5 inches, for example, is at 1.5 inches from the right in right-to-left text.

Functions by Task

Creating Text Tabs

`CTTextTabCreate` (page 140)
Creates and initializes a new text tab object.

Getting Text Tab Data

`CTTextTabGetAlignment` (page 140)
Returns the text alignment of the tab.

`CTTextTabGetLocation` (page 140)
Returns the tab's ruler location.

`CTTextTabGetOptions` (page 141)
Returns the dictionary of attributes associated with the tab.

Getting the Type Identifier

`CTTextTabGetTypeID` (page 141)
Returns the Core Foundation type identifier of the text tab object.

Functions

CTTextTabCreate

Creates and initializes a new text tab object.

```
CTTextTabRef CTTextTabCreate( CTTextAlignment alignment, double location,
CFDictionaryRef options );
```

Parameters

alignment

The tab's alignment. This is used to determine the position of text inside the tab column. This parameter must be set to a valid `CTTextAlignment` (page 112) value or this function returns `NULL`.

location

The tab's ruler location, relative to the back margin.

options

Options to pass in when the tab is created. Currently, the only option available is `kCTTabColumnTerminatorsAttributeName` (page 142). This parameter is optional and can be set to `NULL` if not needed.

Return Value

A reference to a CTTextTab object if the call was successful; otherwise, `NULL`.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTextTab.h`

CTTextTabGetAlignment

Returns the text alignment of the tab.

```
CTTextAlignment CTTextTabGetAlignment( CTTextTabRef tab );
```

Parameters

tab

The tab whose text alignment is obtained.

Return Value

The tab's text alignment value.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTextTab.h`

CTTextTabGetLocation

Returns the tab's ruler location.

```
double CTTextTabGetLocation( CTTextTabRef tab );
```

Parameters

tab

The tab whose location is obtained.

Return Value

The tab's ruler location relative to the back margin.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTextTab.h`

CTTextTabGetOptions

Returns the dictionary of attributes associated with the tab.

```
CFDictionaryRef CTTextTabGetOptions( CTTextTabRef tab );
```

Parameters

tab

The tab whose attributes are obtained.

Return Value

The dictionary of attributes associated with the tab, or if no dictionary is present, `NULL`.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTextTab.h`

CTTextTabGetTypeID

Returns the Core Foundation type identifier of the text tab object.

```
CFTypeID CTTextTabGetTypeID( void );
```

Availability

Available in iOS 3.2 and later.

Declared In

`CTTextTab.h`

Data Types

CTTextTabRef

A reference to a text tab object.

```
typedef const struct __CTTextTab *CTTextTabRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTTextTab.h`

Constants

kCTTabColumnTerminatorsAttributeName

Specifies the terminating character for a tab column.

```
const CFStringRef kCTTabColumnTerminatorsAttributeName;
```

Constants
`kCTTabColumnTerminatorsAttributeName`
Specifies the terminating character for a tab column.

Available in iOS 3.2 and later.

Declared in `CTTextTab.h`.

Discussion
The value associated with this attribute is a CFCharacterSet object. The character set is used to determine the terminating character for a tab column. The tab and newline characters are implied even if they don't exist in the character set. This attribute can be used to implement decimal tabs, for instance. This attribute is optional.

Declared In
`CTTextTab.h`

CTTypesetter Reference

Derived From:	CFType
Framework:	ApplicationServices/CoreText
Declared in	CTTypesetter.h

Overview

The CTTypesetter opaque type represents a typesetter, which performs line layout.

Line layout includes word wrapping, hyphenation, and line breaking in either vertical or horizontal rectangles. A typesetter object takes as input an attributed string and produces a line of typeset glyphs (composed into glyph runs) in a CTLine object. The typesetter performs character-to-glyph encoding, glyph ordering, and positional operations, such as kerning, tracking, and baseline adjustments. If multiline layout is needed, it is performed by a framesetter object, which calls into the typesetter to generate the typeset lines to fill the frame.

A framesetter encapsulates a typesetter and provides a reference to it as a convenience, but a caller may also choose to create a freestanding typesetter.

Functions by Task

Creating a Typesetter

`CTTypesetterCreateWithAttributedString` (page 145)
Creates an immutable typesetter object using an attributed string.

`CTTypesetterCreateWithAttributedStringAndOptions` (page 146)
Creates an immutable typesetter object using an attributed string and a dictionary of options.

Creating Lines

`CTTypesetterCreateLine` (page 144)
Creates an immutable line from the typesetter.

`CTTypesetterCreateLineWithOffset` (page 145)
Creates an immutable line from the typesetter at a specified line offset.

Breaking Lines

`CTTypesetterSuggestLineBreak` (page 148)
Suggests a contextual line breakpoint based on the width provided.

`CTTypesetterSuggestLineBreakWithOffset` (page 148)
Suggests a contextual line breakpoint based on the width provided and the specified offset.

`CTTypesetterSuggestClusterBreak` (page 146)
Suggests a cluster line breakpoint based on the width provided.

`CTTypesetterSuggestClusterBreakWithOffset` (page 147)
Suggests a cluster line breakpoint based on the specified width and line offset.

Getting the Type Identifier

`CTTypesetterGetTypeID` (page 146)
Returns the Core Foundation type identifier of the typesetter object.

Functions

CTTypesetterCreateLine

Creates an immutable line from the typesetter.

```
CTLineRef CTTypesetterCreateLine( CTTypesetterRef typesetter, CFRange stringRange
 );
```

Parameters

typesetter
The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

stringRange
The string range on which the line is based. If the length portion of range is set to `0`, then the typesetter continues to add glyphs to the line until it runs out of characters in the string. The location and length of the range must be within the bounds of the string, or the call will fail.

Return Value
A reference to a CTLine object if the call was successful; otherwise, `NULL`.

Discussion
The resultant line consists of glyphs in the correct visual order, ready to draw. This function is equivalent to `CTTypesetterCreateLineWithOffset` (page 145) with an offset of 0.0.

Availability
Available in iOS 3.2 and later.

Declared In
`CTTypesetter.h`

CTTypesetterCreateLineWithOffset

Creates an immutable line from the typesetter at a specified line offset.

```
CTLineRef CTTypesetterCreateLineWithOffset(CTTypesetterRef typesetter, CFRange
stringRange, double offset);
```

Parameters

typesetter

The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

stringRange

The string range on which the line is based. If the length portion of range is set to `0`, then the typesetter continues to add glyphs to the line until it runs out of characters in the string. The location and length of the range must be within the bounds of the string, or the call will fail.

offset

The line position offset.

Return Value

A reference to a CTLine object if the call was successful; otherwise, `NULL`.

Discussion

The resultant line consists of glyphs in the correct visual order, ready to draw.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTypesetter.h`

CTTypesetterCreateWithAttributedString

Creates an immutable typesetter object using an attributed string.

```
CTTypesetterRef CTTypesetterCreateWithAttributedString( CFAttributedStringRef string
 );
```

Parameters

string

The attributed string to typeset. This parameter must be filled in with a valid CFAttributedString object.

Return Value

A reference to a CTTypesetter object if the call was successful; otherwise, `NULL`.

Discussion

The resultant typesetter can be used to create lines, perform line breaking, and do other contextual analysis based on the characters in the string.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTypesetter.h`

CTTypesetterCreateWithAttributedStringAndOptions

Creates an immutable typesetter object using an attributed string and a dictionary of options.

```
CTTypesetterRef CTTypesetterCreateWithAttributedStringAndOptions(
CFAttributedStringRef string, CFDictionaryRef options );
```

Parameters

string

The attributed string to typeset. This parameter must be filled in with a valid CFAttributedString object.

options

A dictionary of typesetter options, or `NULL` if there are none.

Return Value

A reference to a CTTypesetter object if the call was successful; otherwise, `NULL`.

Discussion

The resultant typesetter can be used to create lines, perform line breaking, and do other contextual analysis based on the characters in the string.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTypesetter.h`

CTTypesetterGetTypeID

Returns the Core Foundation type identifier of the typesetter object.

```
CFTypeID CTTypesetterGetTypeID( void );
```

Availability

Available in iOS 3.2 and later.

Declared In

`CTTypesetter.h`

CTTypesetterSuggestClusterBreak

Suggests a cluster line breakpoint based on the width provided.

```
CFIndex CTTypesetterSuggestClusterBreak( CTTypesetterRef typesetter, CFIndex
startIndex, double width );
```

Parameters

typesetter

The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

startIndex

The starting point for the typographic cluster-break calculations. The break calculations include the character starting at *startIndex*.

width
The requested typographic cluster-break width.

Return Value
A count of the characters from *startIndex* that would cause the cluster break. The value returned can be used to construct a character range for `CTTypesetterCreateLine` (page 144).

Discussion
This cluster break is similar to a character break, except that it does not break apart linguistic clusters. No other contextual analysis is done. This can be used by the caller to implement a different line-breaking scheme, such as hyphenation. A typographic cluster break can also be triggered by a hard-break character in the stream. This function is equivalent to `CTTypesetterSuggestClusterBreakWithOffset` (page 147) with an offset of 0.0.

Availability
Available in iOS 3.2 and later.

Declared In
`CTTypesetter.h`

CTTypesetterSuggestClusterBreakWithOffset

Suggests a cluster line breakpoint based on the specified width and line offset.

```
CFIndex CTTypesetterSuggestClusterBreakWithOffset( CTTypesetterRef typesetter,
CFIndex startIndex, double width, double offset);
```

Parameters
typesetter
The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

startIndex
The starting point for the typographic cluster-break calculations. The break calculations include the character starting at *startIndex*.

width
The requested typographic cluster-break width.

offset
The line offset position.

Return Value
A count of the characters from *startIndex* that would cause the cluster break. The value returned can be used to construct a character range for `CTTypesetterCreateLine` (page 144).

Discussion
This cluster break is similar to a character break, except that it does not break apart linguistic clusters. No other contextual analysis is done. This can be used by the caller to implement a different line-breaking scheme, such as hyphenation. A typographic cluster break can also be triggered by a hard-break character in the stream.

Availability
Available in iOS 3.2 and later.

Declared In
`CTTypesetter.h`

CTTypesetterSuggestLineBreak

Suggests a contextual line breakpoint based on the width provided.

```
CFIndex CTTypesetterSuggestLineBreak( CTTypesetterRef typesetter, CFIndex startIndex,
 double width );
```

Parameters

typesetter

The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

startIndex

The starting point for the line-break calculations. The break calculations include the character starting at *startIndex*.

width

The requested line-break width.

Return Value

A count of the characters from *startIndex* that would cause the line break. The value returned can be used to construct a character range for `CTTypesetterCreateLine` (page 144).

Discussion

The line break can be triggered either by a hard-break character in the stream or by filling the specified width with characters. This function is equivalent to `CTTypesetterSuggestLineBreakWithOffset` (page 148) with an offset of 0.0.

Availability

Available in iOS 3.2 and later.

Declared In

`CTTypesetter.h`

CTTypesetterSuggestLineBreakWithOffset

Suggests a contextual line breakpoint based on the width provided and the specified offset.

```
CFIndex CTTypesetterSuggestLineBreakWithOffset(CTTypesetterRef typesetter, CFIndex
 startIndex, double width, double offset);
```

Parameters

typesetter

The typesetter that creates the line. This parameter is required and cannot be set to `NULL`.

startIndex

The starting point for the line-break calculations. The break calculations include the character starting at *startIndex*.

width

The requested line-break width.

offset

The line position offset.

Return Value

A count of the characters from *startIndex* and *offset* that would cause the line break. The value returned can be used to construct a character range for `CTTypesetterCreateLine` (page 144).

Discussion
The line break can be triggered either by a hard-break character in the stream or by filling the specified width with characters.

Availability
Available in iOS 3.2 and later.

Declared In
`CTTypesetter.h`

Data Types

CTTypesetterRef

A reference to a typesetter object.

```
typedef const struct __CTTypesetter *CTTypesetterRef;
```

Availability
Available in iOS 3.2 and later.

Declared In
`CTTypesetter.h`

Constants

Typesetter Options

These constants control aspects of the typesetter's bidirectional text processing.

```
const CFStringRef kCTTypesetterOptionDisableBidiProcessing;
const CFStringRef kCTTypesetterOptionForcedEmbeddingLevel;
```

Constants

`kCTTypesetterOptionDisableBidiProcessing`

Disables bidirectional processing. Value must be a CFBoolean object. Default value is `false`. Normally, typesetting applies the Unicode Bidirectional Algorithm as described in Unicode Standard Annex #9. If a typesetter is created with this option set to `true`, no directional reordering is performed, and any directional control characters are ignored.

Available in iOS 3.2 and later.

Declared in `CTTypesetter.h`.

`kCTTypesetterOptionForcedEmbeddingLevel`

Specifies the embedding level. Value must be a `CFNumberRef` object. Default is unset. Normally, typesetting applies the Unicode Bidirectional Algorithm as described in Unicode Standard Annex #9. If present, this option specifies the embedding level, and any directional control characters are ignored.

Available in iOS 3.2 and later.

Declared in `CTTypesetter.h`.

Managers

Core Text Utilities Reference

Framework: ApplicationServices/CoreText

Declared in CoreText.h

Overview

This reference document describes miscellaneous symbols that are either used by many different opaque types or apply to Core Text as a whole.

Functions

CTGetCoreTextVersion

Returns the version of the Core Text framework.

```
uint32_t CTGetCoreTextVersion( void );
```

Return Value
The version number. This value is for comparison with the constants listed in "Core Text Framework Version Numbers" (page 154).

Discussion
This function returns a number indicating the version of the Core Text framework. Note that framework version is not always an accurate indicator of feature availability. The recommended way to use this function is first to check that the function pointer is non-null, followed by calling it and comparing its result to a defined constant (or constants). For example, to determine whether the CoreText API is available:

```
if (&CTGetCoreTextVersion != NULL && CTGetCoreTextVersion() >=
kCTVersionNumber10_5) {
 // CoreText API is available
}
```

Availability
Available in iOS 3.2 and later.

Declared In
CoreText.h

Constants

Core Text Framework Version Numbers

Version numbers of the Core Text framework.

```
#define kCTVersionNumber10_5 0x00020000
#define kCTVersionNumber10_5_2 0x00020001
#define kCTVersionNumber10_5_3 0x00020002
#define kCTVersionNumber10_5_5 0x00020003
#define kCTVersionNumber10_6 0x00030000
```

Constants

`kCTVersionNumber10_5`

The Core Text framework version in Mac OS X version 10.5.

Available in iOS 3.2 and later.

Declared in `CoreText.h`.

`kCTVersionNumber10_5_2`

The Core Text framework version in Mac OS X version 10.5.2.

Available in iOS 3.2 and later.

Declared in `CoreText.h`.

`kCTVersionNumber10_5_3`

The Core Text framework version in Mac OS X version 10.5.3.

Available in iOS 3.2 and later.

Declared in `CoreText.h`.

`kCTVersionNumber10_5_5`

The Core Text framework version in Mac OS X version 10.5.5.

Available in iOS 3.2 and later.

Declared in `CoreText.h`.

`kCTVersionNumber10_6`

The Core Text framework version in Mac OS X version 10.6.

Available in iOS 3.2 and later.

Declared in `CoreText.h`.

Declared In

`CoreText.h`

Other References

Core Text String Attributes Reference

Framework: ApplicationServices/CoreText

Declared in CTStringAttributes.h

Overview

This reference document describes the attributes to which Core Text responds when the attributes are placed in a CFAttributedString object.

Constants

String Attribute Name Constants

These constants represent string attribute names.

```
const CFStringRef kCTCharacterShapeAttributeName;
const CFStringRef kCTFontAttributeName;
const CFStringRef kCTKernAttributeName;
const CFStringRef kCTLigatureAttributeName;
const CFStringRef kCTForegroundColorAttributeName;
const CFStringRef kCTForegroundColorFromContextAttributeName;
const CFStringRef kCTParagraphStyleAttributeName;
const CFStringRef kCTStrokeWidthAttributeName;
const CFStringRef kCTStrokeColorAttributeName;
const CFStringRef kCTSuperscriptAttributeName;
const CFStringRef kCTUnderlineColorAttributeName;
const CFStringRef kCTUnderlineStyleAttributeName;
const CFStringRef kCTVerticalFormsAttributeName;
const CFStringRef kCTGlyphInfoAttributeName;
const CFStringRef kCTRunDelegateAttributeName
```

Constants

`kCTCharacterShapeAttributeName`

Controls glyph selection. Value must be a `CFNumberRef` object. Default is value is `0` (disabled). A non-zero value is interpreted as Apple Type Services `kCharacterShapeType` selector `+ 1` (see `<ATS/SFNTLayoutTypes.h>` for selectors). For example, an attribute value of `1` corresponds to `kTraditionalCharactersSelector`.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTFontAttributeName`
The font of the text to which this attribute applies. The value associated with this attribute must be a CTFont object. Default is Helvetica 12.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTKernAttributeName`
The amount to kern the next character. The value associated with this attribute must be a CFNumber float. Default is standard kerning. The kerning attribute indicates how many points the following character should be shifted from its default offset as defined by the current character's font in points: a positive kern indicates a shift farther away from and a negative kern indicates a shift closer to the current character. If this attribute is not present, standard kerning is used. If this attribute is set to `0.0`, no kerning is done at all.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTLigatureAttributeName`
The type of ligatures to use. The value associated with this attribute must be a CFNumber object. Default is an integer value of `1`. The ligature attribute determines what kinds of ligatures should be used when displaying the string. A value of `0` indicates that only ligatures essential for proper rendering of text should be used. A value of `1` indicates that standard ligatures should be used, and `2` indicates that all available ligatures should be used. Which ligatures are standard depends on the script and possibly the font. Arabic text, for example, requires ligatures for many character sequences but has a rich set of additional ligatures that combine characters. English text has no essential ligatures, and typically has only two standard ligatures, those for "fi" and "fl"—all others are considered more advanced or fancy.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTForegroundColorAttributeName`
The foreground color of the text to which this attribute applies. The value associated with this attribute must be a CGColor object. Default value is black.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTForegroundColorFromContextAttributeName`
Sets a foreground color using the context's fill color. Value must be a `CFBooleanRef` object. Default is `false`. The reason this exists is because an `NSAttributedString` object defaults to a black color if no color attribute is set. This forces Core Text to set the color in the context. This attribute allows developers to sidestep this, making Core Text set nothing but font information in the `CGContext`. If set, this attribute also determines the color used by `kCTUnderlineStyleAttributeName` (page 159), in which case it overrides the foreground color.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTParagraphStyleAttributeName`
The paragraph style of the text to which this attribute applies. A paragraph style object is used to specify things like line alignment, tab rulers, writing direction, and so on. Value must be a CTParagraphStyle object. Default is an empty CTParagraphStyle object. See *CTParagraphStyle Reference* for more information.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTStrokeWidthAttributeName`
: The stroke width. Value must be a `CFNumberRef` object. Default value is `0.0`, or no stroke. This attribute, interpreted as a percentage of font point size, controls the text drawing mode: positive values effect drawing with stroke only; negative values are for stroke and fill. A typical value for outlined text is `3.0`.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTStrokeColorAttributeName`
: The stroke color. Value must be a `CGColorRef` object. Default is the foreground color.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTSuperscriptAttributeName`
: Controls vertical text positioning. Value must be a `CFNumberRef` object. Default is integer value `0`. If supported by the specified font, a value of `1` enables superscripting and a value of `-1` enables subscripting.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlineColorAttributeName`
: The underline color. Value must be a `CGColorRef` object. Default is the foreground color.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlineStyleAttributeName`
: The style of underlining, to be applied at render time, for the text to which this attribute applies. Value must be a CFNumber object. Default is `kCTUnderlineStyleNone`. Set a value of something other than `kCTUnderlineStyleNone` to draw an underline. In addition, the constants listed in "CTUnderlineStyleModifiers" (page 161) can be used to modify the look of the underline. The underline color is determined by the text's foreground color.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTVerticalFormsAttributeName`
: The orientation of the glyphs in the text to which this attribute applies. Value must be a CFBoolean object. Default is `False`. A value of `False` indicates that horizontal glyph forms are to be used; `True` indicates that vertical glyph forms are to be used.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTGlyphInfoAttributeName`
: The glyph info object to apply to the text associated with this attribute. Value must be a CTGlyphInfo object. The glyph specified by this CTGlyphInfo object is assigned to the entire attribute range, provided that its contents match the specified base string and that the specified glyph is available in the font specified by `kCTFontAttributeName`. See *CTGlyphInfo Reference* for more information.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTRunDelegateAttributeName`
: The run-delegate object to apply to an attribute range of the string. The value must be a CTRunDelegate object. The run delegate controls such typographic traits as glyph ascent, descent, and width. The values returned by the embedded run delegate apply to each glyph resulting from the text in that range. Because an embedded object is only a display-time modification, you should avoid applying this attribute to a range of text with complex behavior, such as text having a change of writing direction or having combining marks. It is thus recommended you apply this attribute to a range containing the single character U+FFFC. See CTRunDelegate Reference for more information.

  Available in iOS 3.2 and later.

  Declared in `CTStringAttributes.h`.

CTUnderlineStyle

Underline style specifiers.

```
enum{
kCTUnderlineStyleNone = 0x00,
kCTUnderlineStyleSingle = 0x01,
kCTUnderlineStyleThick = 0x02,
kCTUnderlineStyleDouble = 0x09
};
typedef int32_t CTUnderlineStyle;
```

Constants

`kCTUnderlineStyleNone`
: Do not draw an underline.

  Available in iOS 3.2 and later.

  Declared in `CTStringAttributes.h`.

`kCTUnderlineStyleSingle`
: Draw an underline consisting of a single line.

  Available in iOS 3.2 and later.

  Declared in `CTStringAttributes.h`.

`kCTUnderlineStyleThick`
: Draw an underline consisting of a thick line.

  Available in iOS 3.2 and later.

  Declared in `CTStringAttributes.h`.

`kCTUnderlineStyleDouble`
: Draw an underline consisting of a double line.

  Available in iOS 3.2 and later.

  Declared in `CTStringAttributes.h`.

Discussion

These underline type specifiers can be applied to the value set with the `kCTUnderlineStyleAttributeName` (page 159) attribute to control the underline style Core Text uses when rendering the text to which the attribute applies.

CTUnderlineStyleModifiers

Underline style modifiers.

```
enum{
kCTUnderlinePatternSolid = 0x0000,
kCTUnderlinePatternDot = 0x0100,
kCTUnderlinePatternDash = 0x0200,
kCTUnderlinePatternDashDot = 0x0300,
kCTUnderlinePatternDashDotDot = 0x0400
};
typedef int32_t CTUnderlineStyleModifiers;
```

Constants

`kCTUnderlinePatternSolid`
Draw a solid underline.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlinePatternDot`
Draw an underline using a pattern of dots.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlinePatternDash`
Draw an underline using a pattern of dashes.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlinePatternDashDot`
Draw an underline using a pattern of alternating dashes and dots.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

`kCTUnderlinePatternDashDotDot`
Draw an underline using a pattern of a dash followed by two dots.

Available in iOS 3.2 and later.

Declared in `CTStringAttributes.h`.

Discussion

Set these bits with the underline style (see "CTUnderlineStyle" (page 160)) that you set with the `kCTUnderlineStyleAttributeName` (page 159) attribute to modify how the underline will be drawn.

Document Revision History

This table describes the changes to *Core Text Reference Collection*.

Date	Notes
2010-02-25	Moving collection of opaque types to iOS 3.2. Added CTRunDelegate.
2009-11-17	Added link in introduction to companion conceptual document: Core Text Programming Guide.
2009-03-09	Updated for Mac OS X v10.6.
2007-12-04	Changed title from Core Text Framework Reference because it is a subframework of the Application Services framework.
2007-05-09	New document that describes the C API that provides a modern, low-level, high-performance technology for laying out text and handling fonts.

Document Revision History