

---

# Core Text Reference Collection

[Carbon > Text & Fonts](#)



2007-12-04



Apple Inc.  
© 2007 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Carbon, Cocoa, Mac, Mac OS, Macintosh, and QuickDraw are trademarks of Apple Inc., registered in the United States and other countries.

Numbers is a trademark of Apple Inc.

Adobe, Acrobat, and PostScript are trademarks or registered trademarks of Adobe Systems Incorporated in the U.S. and/or other countries.

Helvetica is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

**Introduction**      **Core Text Reference Collection** 7

---

**Part I**      **Opaque Types** 9

---

**Chapter 1**      **CTFont Reference** 11

---

Overview 11  
Functions by Task 11  
Functions 14  
Data Types 40  
Constants 40

**Chapter 2**      **CTFontCollection Reference** 57

---

Overview 57  
Functions by Task 57  
Functions 58  
Data Types 60  
Constants 61

**Chapter 3**      **CTFontDescriptor Reference** 63

---

Overview 63  
Functions by Task 63  
Functions 64  
Data Types 70  
Constants 70

**Chapter 4**      **CTFrame Reference** 79

---

Overview 79  
Functions by Task 79  
Functions 80  
Data Types 83  
Constants 83

**Chapter 5**      **CTFramesetter Reference** 85

---

Overview 85  
Functions by Task 85  
Functions 86

Data Types 87

---

**Chapter 6**      **CTGlyphInfo Reference 89**

---

Overview 89  
Functions by Task 89  
Functions 90  
Data Types 93  
Constants 93

---

**Chapter 7**      **CTLine Reference 95**

---

Overview 95  
Functions by Task 95  
Functions 96  
Data Types 103  
Constants 103

---

**Chapter 8**      **CTParagraphStyle Reference 105**

---

Overview 105  
Functions by Task 105  
Functions 106  
Data Types 108  
Constants 108

---

**Chapter 9**      **CTRun Reference 115**

---

Overview 115  
Functions by Task 115  
Functions 116  
Data Types 123  
Constants 123

---

**Chapter 10**      **CTTextTab Reference 125**

---

Overview 125  
Functions by Task 125  
Functions 126  
Data Types 127  
Constants 128

---

**Chapter 11**      **CTTypesetter Reference 129**

---

Overview 129  
Functions by Task 129

Functions 130  
Data Types 133  
Constants 133

**Part II           Managers 135**

---

**Chapter 12       Core Text Utilities Reference 137**

---

Overview 137  
Functions 137  
Constants 138

**Part III           Other References 139**

---

**Chapter 13       Core Text String Attributes Reference 141**

---

Overview 141  
Constants 141

**Document Revision History 145**

---

**Index 147**

---



# Core Text Reference Collection

---

<b>Framework</b>	/System/Library/Frameworks/ApplicationServices.framework/
<b>Header file directories</b>	/System/Library/Frameworks/ApplicationServices.framework/Headers
<b>Companion guide</b>	Core Text Programming Guide
<b>Declared in</b>	CTFont.h CTFontCollection.h CTFontDescriptor.h CTFontTraits.h CTFrame.h CTFramesetter.h CTGlyphInfo.h CTLine.h CTParagraphStyle.h CTRun.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. It easily fills frames with typeset text and provides access to lines, glyph runs, and individual glyphs. It also enables control of line breaking, paragraph styles, tab settings, and many other aspects of the typesetting process.

The complementary Core Text font technology is designed to handle Unicode fonts natively and comprehensively, giving Carbon developers font technology equivalent to that of Cocoa. It provides automatic font substitution (cascading), font descriptors and collections, easy access to font metrics and glyph data, and many other features.

## INTRODUCTION

Core Text Reference Collection



# Opaque Types

---



# CTFont Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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.

[CTFontCreateWithFontDescriptor](#) (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 36)

Returns the point size of the given font.

[CTFontGetMatrix](#) (page 35)

Returns the transformation matrix of the given font.

[CTFontGetSymbolicTraits](#) (page 37)

Returns the symbolic traits of the given font.

[CTFontCopyTraits](#) (page 22)

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 16)

Returns the Unicode character set of the font.

[CTFontGetStringEncoding](#) (page 37)

Returns the best string encoding for legacy format support.

[CTFontCopySupportedLanguages](#) (page 21)

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 38)  
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 39)  
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 19)

Returns a Core Graphics font reference and attributes.

[CTFontCreateWithGraphicsFont](#) (page 27)

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

[CTFontGetPlatformFont](#) (page 35)

Returns an ATS font reference and attributes.

[CTFontCreateWithPlatformFont](#) (page 28)

Creates a new font reference from an ATS font reference.

[CTFontCreateWithQuickdrawInstance](#) (page 29)

Returns a font reference for the given QuickDraw instance.

## 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.

```
CTypeRef 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 CType.

**Availability**

Available in Mac OS X v10.5 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 [CTFontTableTag](#) (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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 44) 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 `CTFontTableTag` (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 Mac OS X v10.5 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 Mac OS X v10.5 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 43), which represents the four-character axis code as a CFNumber object.

### Availability

Available in Mac OS X v10.5 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 43).

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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,
    uint32_t symTraitValue,
    uint32_t 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 45) 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

`CTFont.h`

**CTFontCreateWithPlatformFont**

Creates a new font reference from an ATS font reference.

```
CTFontRef CTFontCreateWithPlatformFont (
    ATSTFontRef platformFont,
    CGFloat size,
    const CGAffineTransform *matrix,
    CTFontDescriptorRef attributes
);
```

**Parameters***platformFont*

A valid ATSTFontRef object.

*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*

A CTFontDescriptorRef (page 70) containing additional attributes that should be matched. Optional.

**Return Value**

A new font reference for an ATSTFontRef with the specified size, matrix, and additional attributes.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTFont.h

**CTFontCreateWithQuickdrawInstance**

Returns a font reference for the given QuickDraw instance.

```
CTFontRef CTFontCreateWithQuickdrawInstance (
    ConstStr255Param name,
    int16_t identifier,
    uint8_t style,
    CGFloat size
);
```

**Parameters***name*The QuickDraw font name. If zero length, *identifier* must be specified.*identifier*The QuickDraw font identifier. Can be 0, but if so, *name* must be specified.*style*

The QuickDraw font style.

*size*

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

**Return Value**

The best font instance matching the QuickDraw instance information.

**Discussion**

This function is provided for compatibility support between Core Text and clients needing to support QuickDraw-style font references. QuickDraw is a deprecated technology in Mac OS X v10.4 and later.

**Availability**

Available in Mac OS X v10.5 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 determine 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 determine 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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.

```
Boolean 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTFont.h

**CTFontGetPlatformFont**

Returns an ATS font reference and attributes.

```
ATSFonRef CTFontGetPlatformFont (
    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**

An `ATSFontRef` object for the given font reference.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 transformation matrix applied.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

### Declared In

CTFont.h

## CTFontGetTypeID

Returns the type identifier for Core Text font references.

```
CTypeID CTFontGetTypeID (  
    void  
);
```

### Return Value

The identifier for the CTFont opaque type.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontFamilyNameKey`

The name specifier for the family name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontSubFamilyNameKey`

The name specifier for the subfamily name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontStyleNameKey`

The name specifier for the style name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontUniqueNameKey`

The name specifier for the unique name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontFullNameKey`

The name specifier for the full name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVersionNameKey`

The name specifier for the version name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontPostScriptNameKey`

The name specifier for the PostScript name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTrademarkNameKey`

The name specifier for the trademark name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontManufacturerNameKey`

The name specifier for the manufacturer name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontDesignerNameKey`

The name specifier for the designer name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontDescriptionNameKey`

The name specifier for the description name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVendorURLNameKey`

The name specifier for the vendor URL name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontDesignerURLNameKey`

The name specifier for the designer URL name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontLicenseNameKey`

The name specifier for the license name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontLicenseURLNameKey`

The name specifier for the license URL name.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontSampleTextNameKey`

The name specifier for the sample text name string.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontPostScriptCIDNameKey`

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

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisMinimumValueKey`

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisMaximumValueKey`

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisDefaultValueKey`

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontVariationAxisNameKey`

Key to get the localized variation axis name string.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontFeatureTypeNameKey`

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

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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,
      kCTFontMenuItemFontType = 11,
      kCTFontMenuItemMarkFontType = 12,
      kCTFontMenuItemCmdKeyFontType = 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontEmphasizedSystemFontType**

The system font used for emphasis in alerts.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontSmallEmphasizedSystemFontType**

The small system font used for emphasis.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontMiniSystemFontType**

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontMiniEmphasizedSystemFontType**

The miniature system font used for emphasis.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontViewsFontType**

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

**kCTFontApplicationFontType**

The default font for text documents.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontLabelFontType`

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

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemFontType`

The font used for menu titles.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemFontType`

The font used for menu items.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemMarkFontType`

The font used to draw menu item marks.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontMenuItemCmdKeyFontType`

The font used for menu-item command-key equivalents.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontWindowTitleFontType`

The font used for window titles.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontUtilityWindowTitleFontType`

The font used for utility window titles.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontAlertHeaderFontType`

The font used for alert headers.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontSystemDetailFontType`

The standard system font used for details.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontEmphasizedSystemDetailFontType`

The system font used for emphasis in details.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontToolbarFontType`

The font used for labels of toolbar items.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontSmallToolbarFontType`

The small font used for labels of toolbar items.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontPaletteFontType`

The font used in tool palettes.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontToolTipFontType`

The font used for tool tips.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontControlContentFontType`

The font used for contents of user-interface controls.

Available in Mac OS X v10.5 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',
kCTFontTableGlyph     = '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      = 'vmx'
};
typedef uint32_t CTFontTableTag;

```

## Constants

- `kCTFontTableBASE`  
Font table tag for the font baseline.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableCFF`  
Font table tag for a PostScript font program.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableDSIG`  
Font table tag for a digital signature.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableEBDT`  
Font table tag for an embedded bitmap.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableEBLC`  
Font table tag for the embedded bitmap location.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableEBSC`  
Font table tag for embedded bitmap scaling.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableGDEF`  
Font table tag for glyph definition.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableGPOS`  
Font table tag for glyph positioning.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableGSUB`  
Font table tag for glyph substitution.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableJSTF`  
Font table tag for justification.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.

- `kCTFontTableLTSH`  
Font table tag for linear threshold.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableOS2`  
Font table tag for OS/2 and Windows-specific metrics.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTablePCLT`  
Font table tag for PCL 5 data.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableVDMX`  
Font table tag for vertical device metrics.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableVORG`  
Font table tag for vertical origin.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableZapf`  
Font table tag for glyph reference.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableAcnt`  
Font table tag for accent attachment.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableAvar`  
Font table tag for axis variation.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableBdat`  
Font table tag for bitmap data.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableBhed`  
Font table tag for bitmap font header.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.

- `kCTFontTableBLoc`  
Font table tag for bitmap location.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableBsln`  
Font table tag for baseline.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableCmap`  
Font table tag for character-to-glyph mapping.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableCvar`  
Font table tag for control value variation, or CVT variation.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableCvt`  
Font table tag for control value table.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableFdsc`  
Font table tag for font descriptor.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableFeat`  
Font table tag for layout feature.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableFmtx`  
Font table tag for font metrics.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableFpgm`  
Font table tag for font program.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableFvar`  
Font table tag for font variation.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.

- `kCTFontTableGasp`  
Font table tag for grid-fitting/scan-conversion.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableGlyph`  
Font table tag for glyph data.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableGvar`  
Font table tag for glyph variation.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableHdmx`  
Font table tag for horizontal device metrics.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableHead`  
Font table tag for font header.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableHhea`  
Font table tag for horizontal header.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableHmtx`  
Font table tag for horizontal metrics.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableHsty`  
Font table tag for horizontal style.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableJust`  
Font table tag for justification.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.
- `kCTFontTableKern`  
Font table tag for kerning.  
Available in Mac OS X v10.5 and later.  
Declared in `CTFont.h`.

`kCTFontTableLcar`

Font table tag for ligature caret.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableLoca`

Font table tag for index to location.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableMaxp`

Font table tag for maximum profile.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableMort`

Font table tag for morph.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableMorx`

Font table tag for extended morph.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableName`

Font table tag for naming table.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableOpbd`

Font table tag for optical bounds.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTablePost`

Font table tag for PostScript information.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTablePrep`

Font table tag for control value program, 'prep' table.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableProp`

Font table tag for properties.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableTrak`

Font table tag for tracking.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableVhea`

Font table tag for vertical header.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableVmtx`

Font table tag for vertical metrics.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFont.h`.

`kCTFontTableOptionExcludeSynthetic`

The font table excludes synthetic font data.

Available in Mac OS X v10.5 and later.

Declared in `CTFont.h`.

#### Declared In

`CTFont.h`





# CTFontCollection Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTFontCollection.h

**CTFontCollectionGetTypeID**

Returns the type identifier for Core Text font collection references.

```
CTTypeID CTFontCollectionGetTypeID (
    void
);
```

**Return Value**

The identifier for the opaque type CTFontCollection.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTFontCollection.h

## Data Types

**CTFontCollectionRef**

A reference to a font collection.

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

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTFontCollection.h

**CTFontCollectionSortDescriptorsCallback**

The collection sorting callback.

```
typedef CFComparisonResult  
(*CTFontCollectionSortDescriptorsCallback)(CTFontDescriptorRef first,  
CTFontDescriptorRef second, void *refCon);
```

**Discussion**

This callback can be specified to obtain the matching font descriptors of a collection in sorted order. The callback function should return the comparison result of the first descriptor to the second descriptor.

**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 Mac OS X v10.5 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

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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.

```
CTypeRef 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 CType object.

#### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTFontDescriptor.h

**CTFontDescriptorCopyLocalizedAttribute**

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

```
CTypeRef 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 may not be preserved over the round trip from descriptor to font and back to descriptor.

**Availability**

Available in Mac OS X v10.5 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 71)) 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 Mac OS X v10.5 and later.

**Declared In**

CTFontDescriptor.h

**CTFontDescriptorGetTypeID**

Returns the type identifier for Core Text font descriptor references.

```
CTypeID CTFontDescriptorGetTypeID (
    void
);
```

**Return Value**

The identifier for the `CTFontDescriptor` opaque type.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTFontDescriptor.h

## Data Types

### CTFontDescriptorRef

A reference to a CTFontDescriptor object.

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

#### Availability

Available in Mac OS X v10.5 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 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;
```

#### Constants

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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFontDescriptor.h`.

**kCTFontTraitsAttribute**

Key for accessing the dictionary of font traits for stylistic information. See [“Font Traits”](#) (page 74) for the list of font traits. The value associated with this key is a `CFDictionaryRef` object.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 44) 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 Mac OS X v10.5 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 Mac OS X v10.5 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. If you want to receive vertical metrics from a font for vertical rendering, specify `kCTFontVerticalOrientation` (page 73). If unspecified, the font uses its native orientation.

Available in Mac OS X v10.5 and later.

Declared in `CTFontDescriptor.h`.

#### Declared In

`CTFontDescriptor.h`

## Font Orientation Constants

Specifies the intended rendering orientation of the font for obtaining glyph metrics.

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

#### Constants

`kCTFontDefaultOrientation`

The native orientation of the font.

Available in Mac OS X v10.5 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontHorizontalOrientation`

Specifies horizontal orientation.

Available in Mac OS X v10.5 and later.

Declared in `CTFontDescriptor.h`.

`kCTFontVerticalOrientation`

Specifies vertical orientation.

Available in Mac OS X v10.5 and later.

Declared in `CTFontDescriptor.h`.

#### 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

#### 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

**Declared In**

`CTFontTraits.h`

**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 74).

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

`kCTFontBoldTrait`

The font typestyle is boldface. Additional detail is available via [kCTFontWeightTrait](#) (page 74).

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

`kCTFontExpandedTrait`

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

Available in Mac OS X v10.5 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 74).

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

`kCTFontVerticalTrait`

The font uses vertical glyph variants and metrics.

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

`kCTFontUIOptimizedTrait`

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

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

`kCTFontClassMaskTrait`

Mask for the font class.

Available in Mac OS X v10.5 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.

### Declared In

`CTFontTraits.h`

## 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

**kCTFontModernSerifsClass**

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

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

**kCTFontClarendonSerifsClass**

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

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

**kCTFontSansSerifClass**

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

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

**kCTFontOrnamentsClass**

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

Available in Mac OS X v10.5 and later.

Declared in `CTFontTraits.h`.

**kCTFontScriptsClass**

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

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 75) and can be obtained via `kCTFontClassMaskTrait` (page 76).

**Declared In**

`CTFontTraits.h`

# CTFrame Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 82)

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

[CTFrameGetVisibleStringRange](#) (page 83)

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

[CTFrameGetPath](#) (page 82)

Returns the path used to create the frame.

[CTFrameGetFrameAttributes](#) (page 80)

Returns the frame attributes used to create the frame.

### Getting Lines

[CTFrameGetLines](#) (page 81)

Returns an array of lines stored in the frame.

[CTFrameGetLineOrigins](#) (page 81)

Copies a range of line origins for a frame.

## Drawing the Frame

[CTFrameDraw](#) (page 80)

Draws an entire frame into a context.

## Getting the Type Identifier

[CTFrameGetTypeID](#) (page 82)

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 Mac OS X v10.5 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 Mac OS X v10.5 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 81), 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTFrame.h`.

`kCTFrameProgressionRightToLeft`

Lines are stacked right to left for vertical text.

Available in Mac OS X v10.5 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 83) constant. The default is `kCTFrameProgressionTopToBottom`.

Available in Mac OS X v10.5 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

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 86)  
Creates an immutable framesetter object from an attributed string.

### Creating Frames

[CTFramesetterCreateFrame](#) (page 86)  
Creates an immutable frame using a framesetter.

[CTFramesetterGetTypeSetter](#) (page 87)  
Returns the typesetter object being used by the framesetter.

### Getting the Type Identifier

[CTFramesetterGetTypeID](#) (page 87)  
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 Mac OS X v10.5 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 86) call.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTFramesetter.h

**CTFramesetterGetTypeID**

Returns the Core Foundation type identifier of the framesetter object.

```
CTTypeID CTFramesetterGetTypeID( void );
```

**Availability**

Available in Mac OS X v10.5 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 CTypesetter 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTFramesetter.h



# CTGlyphInfo Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 92)

Returns the Core Foundation type identifier of the glyph info object

### Creating GlyphInfo Objects

[CTGlyphInfoCreateWithGlyphName](#) (page 91)

Creates an immutable glyph info object with a glyph name.

[CTGlyphInfoCreateWithGlyph](#) (page 90)

Creates an immutable glyph info object with a glyph index.

[CTGlyphInfoCreateWithCharacterIdentifier](#) (page 90)

Creates an immutable glyph info object with a character identifier.

### Getting GlyphInfo Data

[CTGlyphInfoGetGlyphName](#) (page 92)

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

[CTGlyphInfoGetCharacterIdentifier](#) (page 92)

Gets the character identifier for a glyph info object.

[CTGlyphInfoGetCharacterCollection](#) (page 91)

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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 93).**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

#### Declared In

CTGlyphInfo.h

### CTGlyphInfoGetTypeID

Returns the Core Foundation type identifier of the glyph info object

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

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CTGlyphInfo.h

## Data Types

### CTGlyphInfoRef

A reference to a glyph info object.

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

#### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in CTGlyphInfo.h.

kCTAdobeCNS1CharacterCollection

The Adobe-CNS1 mapping.

Available in Mac OS X v10.5 and later.

Declared in CTGlyphInfo.h.

kCTAdobeGB1CharacterCollection

The Adobe-GB1 mapping.

Available in Mac OS X v10.5 and later.

Declared in CTGlyphInfo.h.

kCTAdobeJapan1CharacterCollection

The Adobe-Japan1 mapping.

Available in Mac OS X v10.5 and later.

Declared in CTGlyphInfo.h.

kCTAdobeJapan2CharacterCollection

**The Adobe-Japan2 mapping.**

**Available in Mac OS X v10.5 and later.**

**Declared in** CTGlyphInfo.h.

kCTAdobeKorea1CharacterCollection

**The Adobe-Korea1 mapping.**

**Available in Mac OS X v10.5 and later.**

**Declared in** CTGlyphInfo.h.

**Declared In**

CTGlyphInfo.h

# CTLine Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 97)

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

[CTLineCreateTruncatedLine](#) (page 97)

Creates a truncated line from an existing line.

[CTLineCreateJustifiedLine](#) (page 96)

Creates a justified line from an existing line.

### Drawing the Line

[CTLineDraw](#) (page 98)

Draws a complete line.

### Getting Line Data

[CTLineGetGlyphCount](#) (page 98)

Returns the total glyph count for the line object.

[CTLineGetGlyphRuns](#) (page 99)

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

[CTLineGetStringRange](#) (page 101)

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

[CTLineGetPenOffsetForFlush](#) (page 100)

Gets the pen offset required to draw flush text.

## Measuring Lines

[CTLineGetImageBounds](#) (page 99)

Calculates the image bounds for a line.

[CTLineGetTypographicBounds](#) (page 102)

Calculates the typographic bounds of a line.

[CTLineGetTrailingWhitespaceWidth](#) (page 102)

Returns the trailing whitespace width for a line.

## Getting Line Positioning

[CTLineGetStringIndexForPosition](#) (page 101)

Performs hit testing.

[CTLineGetOffsetForStringIndex](#) (page 100)

Determines the graphical offset or offsets for a string index.

## Getting the Type Identifier

[CTLineGetTypeID](#) (page 102)

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 Mac OS X v10.5 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 103) 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 101). 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTLine.h

**CTLineGetTypeID**

Returns the Core Foundation type identifier of the line object.

```
CTypeID CTLineGetTypeID( void );
```

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTLine.h

## Data Types

**CTLineRef**

A reference to a line object.

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

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTLine.h

## Constants

**CTLineTruncationType**

Truncation types required by the [CTLineCreateTruncatedLine](#) (page 97) 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 Mac OS X v10.5 and later.

Declared in CTLine.h.

kCTLineTruncationEnd

Truncate the end of the line, leaving the start portion visible.

Available in Mac OS X v10.5 and later.

Declared in CTLine.h.

kCTLineTruncationMiddle

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

Available in Mac OS X v10.5 and later.

Declared in `CTLine.h`.

**Declared In**

`CTLine.h`



# CTParagraphStyle Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 106)

Creates an immutable paragraph style.

[CTParagraphStyleCreateCopy](#) (page 106)

Creates an immutable copy of a paragraph style.

### Getting the Value of a Style Specifier

[CTParagraphStyleGetValueForSpecifier](#) (page 107)

Obtains the current value for a single setting specifier.

### Getting the Type Identifier

[CTParagraphStyleGetTypeID](#) (page 107)

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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTParagraphStyle.h

**CTParagraphStyleGetTypeID**

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

```
CTypeID CTParagraphStyleGetTypeID( void );
```

**Availability**

Available in Mac OS X v10.5 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 111) 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 Mac OS X v10.5 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 111) 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 Mac OS X v10.5 and later.

#### Declared In

`CTParagraphStyle.h`

### CTParagraphStyleRef

A reference to a Core Text paragraph style.

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

#### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

kCTRightTextAlignment

Text is visually right aligned.

Available in Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

kCTCenterTextAlignment

Text is visually center aligned.

Available in Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

kCTJustifiedTextAlignment

Text is fully justified. The last line in a paragraph is naturally aligned.

Available in Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

kCTNaturalTextAlignment

Text uses the natural alignment of the text's script.

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByCharWrapping`

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

Available in Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

`kCTLineBreakByClipping`

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

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in CTParagraphStyle.h.

kCTWritingDirectionLeftToRight

The writing direction is left to right.

Available in Mac OS X v10.5 and later.

Declared in CTParagraphStyle.h.

kCTWritingDirectionRightToLeft

The writing direction is right to left.

Available in Mac OS X v10.5 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 108). Default: [kCTNaturalTextAlignment](#) (page 109). Application: CTFramesetter.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 127). Default: 12 left-aligned tabs, spaced by 28.0 points. Application: CTFramesetter, CTTypesetter.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 109). Default: `kCTLineBreakByWordWrapping` (page 110). Application: `CTFramesetter`

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierBaseWritingDirection`

The base writing direction of the lines. Type: `CTWritingDirection` (page 110). Default: `kCTWritingDirectionNatural` (page 111). Application: `CTFramesetter`, `CTTypesetter`.

Available in Mac OS X v10.5 and later.

Declared in `CTParagraphStyle.h`.

`kCTParagraphStyleSpecifierCount`

The number of style specifiers. The purpose is to simplify validation of style specifiers

Available in Mac OS X v10.5 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

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 glyph 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 117)

Gets the glyph count for the run.

[CTRunGetAttributes](#) (page 117)

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

[CTRunGetStatus](#) (page 120)

Returns the run's status.

[CTRunGetGlyphsPtr](#) (page 118)

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

[CTRunGetGlyphs](#) (page 117)

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

[CTRunGetPositionsPtr](#) (page 119)

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

[CTRunGetPositions](#) (page 119)

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

[CTRunGetStringIndicesPtr](#) (page 121)

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

[CTRunGetStringIndices](#) (page 120)

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

[CTRunGetStringRange](#) (page 121)

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

## Measuring the Glyph Run

[CTRunGetTypographicBounds](#) (page 122)

Gets the typographic bounds of the run.

[CTRunGetImageBounds](#) (page 118)

Calculates the image bounds for a glyph range.

## Drawing the Glyph Run

[CTRunDraw](#) (page 116)

Draws a complete run or part of one.

[CTRunGetTextMatrix](#) (page 122)

Returns the text matrix needed to draw this run.

## Getting the Type Identifier

[CTRunGetTypeID](#) (page 122)

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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 117). 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 117). 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 119) to fetch the glyph positions.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 117). 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 120) to fetch the indices.

**Availability**

Available in Mac OS X v10.5 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, or if *run* is invalid, an empty range.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

### Declared In

`CTRun.h`

## CTRunGetTypeID

Returns the Core Foundation type identifier of the run object.

```
CFTypeID CTRunGetTypeID( void );
```

### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

`CTRun.h`

## Data Types

**CTRunRef**

A reference to a run object.

```
typedef const struct __CTRun *CTRunRef;
```

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`CTRun.h`

## Constants

**CTRunStatus**

A bitfield passed back by the [CTRunGetStatus](#) (page 120) 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 Mac OS X v10.5 and later.

Declared in `CTRun.h`.

`kCTRunStatusRightToLeft`

The run proceeds from right to left.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTRun.h`.

**Declared In**

`CTRun.h`

# CTTextTab Reference

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 126)  
Creates and initializes a new text tab object.

### Getting Text Tab Data

[CTTextTabGetAlignment](#) (page 126)  
Returns the text alignment of the tab.

[CTTextTabGetLocation](#) (page 126)  
Returns the tab's ruler location.

[CTTextTabGetOptions](#) (page 127)  
Returns the dictionary of attributes associated with the tab.

### Getting the Type Identifier

[CTTextTabGetTypeID](#) (page 127)  
Returns the Core Foundation type identifier of the text tab object.

## Functions

### CTTextTabCreate

Creates and initializes a new text tab object.

```
CTTextTabRef CTTextTabCreate( CTextAlignment 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 108) 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 128). 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 Mac OS X v10.5 and later.

#### Declared In

`CTTextTab.h`

### CTTextTabGetAlignment

Returns the text alignment of the tab.

```
CTextAlignment CTTextTabGetAlignment( CTextTabRef tab );
```

#### Parameters

*tab*

The tab whose text alignment is obtained.

#### Return Value

The tab's text alignment value.

#### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CTTextTab.h

**CTTextTabGetTypeID**

Returns the Core Foundation type identifier of the text tab object.

```
CFTypeID CTTextTabGetTypeID( void );
```

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTTextTab.h

## Data Types

**CTTextTabRef**

A reference to a text tab object.

```
typedef const struct __CTTextTab *CTTextTabRef;
```

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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

---

<b>Derived From:</b>	CType
<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 130)

Creates an immutable typesetter object using an attributed string.

[CTTypesetterCreateWithAttributedStringAndOptions](#) (page 131)

Creates an immutable typesetter object using an attributed string and a dictionary of options.

### Creating Lines

[CTTypesetterCreateLine](#) (page 130)

Creates an immutable line from the typesetter.

## Breaking Lines

[CTTypesetterSuggestLineBreak](#) (page 132)

Suggests a contextual line breakpoint based on the width provided.

[CTTypesetterSuggestClusterBreak](#) (page 132)

Suggests a cluster line breakpoint based on the width provided.

## Getting the Type Identifier

[CTTypesetterGetTypeID](#) (page 131)

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.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

`CTTypesetter.h`

### CTTypesetterCreateWithAttributedString

Creates an immutable typesetter object using an attributed string.

```
CTTypesetterRef CTypesetterCreateWithAttributedString( 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 CTypesetter 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 Mac OS X v10.5 and later.

**Declared In**

CTypesetter.h

**CTypesetterCreateWithAttributedStringAndOptions**

Creates an immutable typesetter object using an attributed string and a dictionary of options.

```
CTypesetterRef CTypesetterCreateWithAttributedStringAndOptions(
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 CTypesetter 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 Mac OS X v10.5 and later.

**Declared In**

CTypesetter.h

**CTypesetterGetTypeID**

Returns the Core Foundation type identifier of the typesetter object.

```
CFTypeID CTypesetterGetTypeID( void );
```

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CTypesetter.h

**CTypesetterSuggestClusterBreak**

Suggests a cluster line breakpoint based on the width provided.

```
CFIndex CTypesetterSuggestClusterBreak( CTypesetterRef 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 [CTypesetterCreateLine](#) (page 130).

**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 Mac OS X v10.5 and later.

**Declared In**

CTypesetter.h

**CTypesetterSuggestLineBreak**

Suggests a contextual line breakpoint based on the width provided.

```
CFIndex CTypesetterSuggestLineBreak( CTypesetterRef 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 130).

#### 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 Mac OS X v10.5 and later.

#### Declared In

CTTypesetter.h

## Data Types

### CTTypesetterRef

A reference to a typesetter object.

```
typedef const struct __CTTypesetter *CTTypesetterRef;
```

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CTTypesetter.h

## Constants

### kCTTypesetterOptionDisableBidiProcessing

Disables bidirectional processing.

```
const CFStringRef kCTTypesetterOptionDisableBidiProcessing;
```

#### Constants

kCTTypesetterOptionDisableBidiProcessing

Disables bidirectional processing.

Available in Mac OS X v10.5 and later.

Declared in CTTypesetter.h.

**Discussion**

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.

**Declared In**

`CTTypesetter.h`

# Managers

---





# Core Text Utilities Reference

---

<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 138).

#### 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 Mac OS X v10.5 and later.

#### Declared In

CoreText.h

## Constants

### Core Text Framework Version Numbers

Version numbers of the Core Text framework.

```
#define kCTVersionNumber10_5 0x00020000
```

#### Constants

`kCTVersionNumber10_5`

The Core Text framework version in Mac OS X version 10.5.

Available in Mac OS X v10.5 and later.

Declared in `CoreText.h`.

#### Declared In

`CoreText.h`

# Other References

---



# Core Text String Attributes Reference

---

<b>Framework:</b>	ApplicationServices/CoreText
<b>Declared in</b>	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 kCTFontAttributeName;
const CFStringRef kCTKernAttributeName;
const CFStringRef kCTLigatureAttributeName;
const CFStringRef kCTForegroundColorAttributeName;
const CFStringRef kCTParagraphStyleAttributeName;
const CFStringRef kCTUnderlineStyleAttributeName;
const CFStringRef kCTVerticalFormsAttributeName;
const CFStringRef kCTGlyphInfoAttributeName;
```

#### Constants

`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 Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

`kCTKernAttributeName`

The amount to kern the next character. The value associated with this attribute must be a `CGFloat`. 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 143) can be used to modify the look of the underline. The underline color is determined by the text's foreground color.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

**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 Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlineStyleSingle

Draw an underline consisting of a single line.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlineStyleThick

Draw an underline consisting of a thick line.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlineStyleDouble

Draw an underline consisting of a double line.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

### Discussion

These underline type specifiers can be applied to the value set with the [kCTUnderlineStyleAttributeName](#) (page 142) attribute to control the underline style Core Text uses when rendering the text to which the attribute applies.

### Declared In

`CTStringAttributes.h`

## 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 Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlinePatternDot

Draw an underline using a pattern of dots.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlinePatternDash

Draw an underline using a pattern of dashes.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlinePatternDashDot

Draw an underline using a pattern of alternating dashes and dots.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

kCTUnderlinePatternDashDotDot

Draw an underline using a pattern of a dash followed by two dots.

Available in Mac OS X v10.5 and later.

Declared in `CTStringAttributes.h`.

**Discussion**

Set these bits with the underline style (see “[CTUnderlineStyle](#)” (page 143)) that you set with the [kCTUnderlineStyleAttributeName](#) (page 142) attribute to modify how the underline will be drawn.

**Declared In**

`CTStringAttributes.h`



# Document Revision History

---

This table describes the changes to *Core Text Reference Collection*.

Date	Notes
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.

**REVISION HISTORY**

Document Revision History

# Index

---

## C

---

Core Text Framework Version Numbers [138](#)

CTCharacterCollection [93](#)

CTFontCollectionCreateCopyWithFontDescriptors  
function [58](#)

CTFontCollectionCreateFromAvailableFonts  
function [58](#)

CTFontCollectionCreateMatchingFontDescriptors  
function [59](#)

CTFontCollectionCreateMatchingFontDescriptors-  
SortedWithCallback function [59](#)

CTFontCollectionCreateWithFontDescriptors  
function [59](#)

CTFontCollectionGetTypeID function [60](#)

CTFontCollectionRef data type [60](#)

CTFontCollectionSortDescriptorsCallback data  
type [61](#)

CTFontCopyAttribute function [14](#)

CTFontCopyAvailableTables function [15](#)

CTFontCopyCharacterSet function [16](#)

CTFontCopyDisplayName function [16](#)

CTFontCopyFamilyName function [16](#)

CTFontCopyFeatures function [17](#)

CTFontCopyFeatureSettings function [17](#)

CTFontCopyFontDescriptor function [18](#)

CTFontCopyFullName function [18](#)

CTFontCopyGraphicsFont function [19](#)

CTFontCopyLocalizedName function [19](#)

CTFontCopyName function [20](#)

CTFontCopyPostScriptName function [20](#)

CTFontCopySupportedLanguages function [21](#)

CTFontCopyTable function [21](#)

CTFontCopyTraits function [22](#)

CTFontCopyVariation function [22](#)

CTFontCopyVariationAxes function [22](#)

CTFontCreateCopyWithAttributes function [23](#)

CTFontCreateCopyWithFamily function [24](#)

CTFontCreateCopyWithSymbolicTraits function [24](#)

CTFontCreateForString function [25](#)

CTFontCreatePathForGlyph function [25](#)

CTFontCreateUIFontForLanguage function [26](#)

CTFontCreateWithFontDescriptor function [27](#)

CTFontCreateWithGraphicsFont function [27](#)

CTFontCreateWithName function [28](#)

CTFontCreateWithPlatformFont function [28](#)

CTFontCreateWithQuickdrawInstance function [29](#)

CTFontDescriptorCopyAttribute function [64](#)

CTFontDescriptorCopyAttributes function [64](#)

CTFontDescriptorCopyLocalizedAttribute function  
[65](#)

CTFontDescriptorCreateCopyWithAttributes  
function [66](#)

CTFontDescriptorCreateCopyWithFeature function  
[66](#)

CTFontDescriptorCreateCopyWithVariation  
function [67](#)

CTFontDescriptorCreateMatchingFontDescriptor  
function [67](#)

CTFontDescriptorCreateMatchingFontDescriptors  
function [68](#)

CTFontDescriptorCreateWithAttributes function  
[68](#)

CTFontDescriptorCreateWithNameAndSize function  
[69](#)

CTFontDescriptorGetTypeID function [69](#)

CTFontDescriptorRef data type [70](#)

CTFontGetAdvancesForGlyphs function [30](#)

CTFontGetAscent function [30](#)

CTFontGetBoundingBox function [31](#)

CTFontGetBoundingBoxRectsForGlyphs function [31](#)

CTFontGetCapHeight function [32](#)

CTFontGetDescent function [32](#)

CTFontGetGlyphCount function [33](#)

CTFontGetGlyphsForCharacters function [33](#)

CTFontGetGlyphWithName function [34](#)

CTFontGetLeading function [34](#)

CTFontGetMatrix function [35](#)

CTFontGetPlatformFont function [35](#)

CTFontGetSize function [36](#)

CTFontGetSlantAngle function [36](#)

CTFontGetStringEncoding function [37](#)

CTFontGetSymbolicTraits function [37](#)

- CTFontGetTypeID [function 37](#)
  - CTFontGetUnderlinePosition [function 38](#)
  - CTFontGetUnderlineThickness [function 38](#)
  - CTFontGetUnitsPerEm [function 38](#)
  - CTFontGetVerticalTranslationsForGlyphs [function 39](#)
  - CTFontGetXHeight [function 39](#)
  - CTFontRef [data type 40](#)
  - CTFrameDraw [function 80](#)
  - CTFrameGetFrameAttributes [function 80](#)
  - CTFrameGetLineOrigins [function 81](#)
  - CTFrameGetLines [function 81](#)
  - CTFrameGetPath [function 82](#)
  - CTFrameGetStringRange [function 82](#)
  - CTFrameGetTypeID [function 82](#)
  - CTFrameGetVisibleStringRange [function 83](#)
  - CTFrameProgression 83**
  - CTFrameRef [structure 83](#)
  - CTFramesetterCreateFrame [function 86](#)
  - CTFramesetterCreateWithAttributedString [function 86](#)
  - CTFramesetterGetTypeID [function 87](#)
  - CTFramesetterGetTypeSetter [function 87](#)
  - CTFramesetterRef [structure 87](#)
  - CTGetCoreTextVersion [function 137](#)
  - CTGlyphInfoCreateWithCharacterIdentifier [function 90](#)
  - CTGlyphInfoCreateWithGlyph [function 90](#)
  - CTGlyphInfoCreateWithGlyphName [function 91](#)
  - CTGlyphInfoGetCharacterCollection [function 91](#)
  - CTGlyphInfoGetCharacterIdentifier [function 92](#)
  - CTGlyphInfoGetGlyphName [function 92](#)
  - CTGlyphInfoGetTypeID [function 92](#)
  - CTGlyphInfoRef [structure 93](#)
  - CTLineBreakMode 109**
  - CTLineCreateJustifiedLine [function 96](#)
  - CTLineCreateTruncatedLine [function 97](#)
  - CTLineCreateWithAttributedString [function 97](#)
  - CTLineDraw [function 98](#)
  - CTLineGetGlyphCount [function 98](#)
  - CTLineGetGlyphRuns [function 99](#)
  - CTLineGetImageBounds [function 99](#)
  - CTLineGetOffsetForStringIndex [function 100](#)
  - CTLineGetPenOffsetForFlush [function 100](#)
  - CTLineGetStringIndexForPosition [function 101](#)
  - CTLineGetStringRange [function 101](#)
  - CTLineGetTrailingWhitespaceWidth [function 102](#)
  - CTLineGetTypeID [function 102](#)
  - CTLineGetTypographicBounds [function 102](#)
  - CTLineRef [structure 103](#)
  - CTLineTruncationType 103**
  - CTParagraphStyleCreate [function 106](#)
  - CTParagraphStyleCreateCopy [function 106](#)
  - CTParagraphStyleGetTypeID [function 107](#)
  - CTParagraphStyleGetValueForSpecifier [function 107](#)
  - CTParagraphStyleRef [data type 108](#)
  - CTParagraphStyleSetting [structure 108](#)
  - CTParagraphStyleSpecifier 111**
  - CTRunDraw [function 116](#)
  - CTRunGetAttributes [function 117](#)
  - CTRunGetGlyphCount [function 117](#)
  - CTRunGetGlyphs [function 117](#)
  - CTRunGetGlyphsPtr [function 118](#)
  - CTRunGetImageBounds [function 118](#)
  - CTRunGetPositions [function 119](#)
  - CTRunGetPositionsPtr [function 119](#)
  - CTRunGetStatus [function 120](#)
  - CTRunGetStringIndices [function 120](#)
  - CTRunGetStringIndicesPtr [function 121](#)
  - CTRunGetStringRange [function 121](#)
  - CTRunGetTextMatrix [function 122](#)
  - CTRunGetTypeID [function 122](#)
  - CTRunGetTypographicBounds [function 122](#)
  - CTRunRef [structure 123](#)
  - CTRunStatus 123**
  - CTTextAlignment 108**
  - CTTextTabCreate [function 126](#)
  - CTTextTabGetAlignment [function 126](#)
  - CTTextTabGetLocation [function 126](#)
  - CTTextTabGetOptions [function 127](#)
  - CTTextTabGetTypeID [function 127](#)
  - CTTextTabRef [structure 127](#)
  - CTTypesetterCreateLine [function 130](#)
  - CTTypesetterCreateWithAttributedString [function 130](#)
  - CTTypesetterCreateWithAttributedStringAndOptions [function 131](#)
  - CTTypesetterGetTypeID [function 131](#)
  - CTTypesetterRef [structure 133](#)
  - CTTypesetterSuggestClusterBreak [function 132](#)
  - CTTypesetterSuggestLineBreak [function 132](#)
  - CTUnderlineStyle 143**
  - CTUnderlineStyleModifiers 143**
  - CTWritingDirection [110](#)
- ## F
- 
- Font Attribute Constants [70](#)
  - Font Class Mask Shift Constants [74](#)
  - Font Feature Constants [43](#)
  - Font Orientation Constants [73](#)
  - Font Stylistic Class Constants [76](#)
  - Font Symbolic Traits Constants [75](#)
  - Font Table Option Constants [55](#)

Font Table Tag Constants 48  
 Font Trait Constants 74  
 Font Variation Axis Dictionary Keys 43

## K

kCTAdobeCNS1CharacterCollection constant 93  
 kCTAdobeGB1CharacterCollection constant 93  
 kCTAdobeJapan1CharacterCollection constant 93  
 kCTAdobeJapan2CharacterCollection constant 94  
 kCTAdobeKorea1CharacterCollection constant 94  
 kCTCenterTextAlignment constant 109  
 kCTFontAlertHeaderFontType constant 47  
 kCTFontApplicationFontType constant 46  
 kCTFontAttributeName constant 141  
 kCTFontBaselineAdjustAttribute constant 72  
 kCTFontBoldTrait constant 75  
 kCTFontCascadelistAttribute constant 71  
 kCTFontCharacterSetAttribute constant 72  
 kCTFontClarendonSerifsClass constant 77  
 kCTFontClassMaskShift constant 75  
 kCTFontClassMaskTrait constant 76  
 kCTFontCollectionRemoveDuplicatesOption 61  
 kCTFontCollectionRemoveDuplicatesOption constant 61  
 kCTFontCondensedTrait constant 75  
 kCTFontControlContentFontType constant 48  
 kCTFontCopyrightNameKey constant 41  
 kCTFontDefaultOrientation constant 73  
 kCTFontDescriptionNameKey constant 42  
 kCTFontDesignerNameKey constant 42  
 kCTFontDesignerURLNameKey constant 42  
 kCTFontDisplayNameAttribute constant 71  
 kCTFontEmphasizedSystemDetailFontType constant 48  
 kCTFontEmphasizedSystemFontType constant 46  
 kCTFontExpandedTrait constant 75  
 kCTFontFamilyNameAttribute constant 71  
 kCTFontFamilyNameKey constant 41  
 kCTFontFeaturesAttribute constant 72  
 kCTFontFeatureSelectorDefaultKey constant 44  
 kCTFontFeatureSelectorIdentifierKey constant 44  
 kCTFontFeatureSelectorNameKey constant 44  
 kCTFontFeatureSelectorSettingKey constant 45  
 kCTFontFeatureSettingsAttribute constant 72  
 kCTFontFeatureTypeExclusiveKey constant 44  
 kCTFontFeatureTypeIdentifierKey constant 44  
 kCTFontFeatureTypeNameKey constant 44  
 kCTFontFeatureTypeSelectorsKey constant 44  
 kCTFontFixedAdvanceAttribute constant 73  
 kCTFontFreeformSerifsClass constant 77  
 kCTFontFullNameKey constant 41  
 kCTFontHorizontalOrientation constant 73  
 kCTFontItalicTrait constant 75  
 kCTFontLabelFontType constant 47  
 kCTFontLanguagesAttribute constant 72  
 kCTFontLicenseNameKey constant 42  
 kCTFontLicenseURLNameKey constant 42  
 kCTFontMacintoshEncodingsAttribute constant 72  
 kCTFontManufacturerNameKey constant 42  
 kCTFontMatrixAttribute constant 71  
 kCTFontMenuItemCmdKeyFontType constant 47  
 kCTFontMenuItemFontType constant 47  
 kCTFontMenuItemMarkFontType constant 47  
 kCTFontMenuItemTitleFontType constant 47  
 kCTFontMessageFontType constant 48  
 kCTFontMiniEmphasizedSystemFontType constant 46  
 kCTFontMiniSystemFontType constant 46  
 kCTFontModernSerifsClass constant 77  
 kCTFontMonoSpaceTrait constant 75  
 kCTFontNameAttribute constant 70  
 kCTFontNoFontType constant 45  
 kCTFontOldStyleSerifsClass constant 76  
 kCTFontOrientationAttribute constant 73  
 kCTFontOrnamentsClass constant 77  
 kCTFontPaletteFontType constant 48  
 kCTFontPostScriptCIDNameKey constant 43  
 kCTFontPostScriptNameKey constant 42  
 kCTFontPushButtonFontType constant 47  
 kCTFontSampleTextNameKey constant 42  
 kCTFontSansSerifClass constant 77  
 kCTFontScriptsClass constant 77  
 kCTFontSizeAttribute constant 71  
 kCTFontSlabSerifsClass constant 77  
 kCTFontSlantTrait constant 74  
 kCTFontSmallEmphasizedSystemFontType constant 46  
 kCTFontSmallSystemFontType constant 46  
 kCTFontSmallToolbarFontType constant 48  
 kCTFontStyleNameAttribute constant 71  
 kCTFontStyleNameKey constant 41  
 kCTFontSubFamilyNameKey constant 41  
 kCTFontSymbolicClass constant 77  
 kCTFontSymbolicTrait constant 74  
 kCTFontSystemDetailFontType constant 47  
 kCTFontSystemFontType constant 46  
 kCTFontTableAcnt constant 51  
 kCTFontTableAvar constant 51  
 kCTFontTableBASE constant 50  
 kCTFontTableBdat constant 51  
 kCTFontTableBhed constant 51  
 kCTFontTableBloc constant 52  
 kCTFontTableBsln constant 52

- kCTFontTableCFF constant 50
- kCTFontTableCmap constant 52
- kCTFontTableCvar constant 52
- kCTFontTableCvt constant 52
- kCTFontTableDSIG constant 50
- kCTFontTableEBDT constant 50
- kCTFontTableEBLC constant 50
- kCTFontTableEBSC constant 50
- kCTFontTableFdsc constant 52
- kCTFontTableFeat constant 52
- kCTFontTableFmtx constant 52
- kCTFontTableFpgm constant 52
- kCTFontTableFvar constant 52
- kCTFontTableGasp constant 53
- kCTFontTableGDEF constant 50
- kCTFontTableGlyf constant 53
- kCTFontTableGPOS constant 50
- kCTFontTableGSUB constant 50
- kCTFontTableGvar constant 53
- kCTFontTableHdmx constant 53
- kCTFontTableHead constant 53
- kCTFontTableHhea constant 53
- kCTFontTableHmtx constant 53
- kCTFontTableHsty constant 53
- kCTFontTableJSTF constant 50
- kCTFontTableJust constant 53
- kCTFontTableKern constant 53
- kCTFontTableLcar constant 54
- kCTFontTableLoca constant 54
- kCTFontTableLTSH constant 51
- kCTFontTableMaxp constant 54
- kCTFontTableMort constant 54
- kCTFontTableMorx constant 54
- kCTFontTableName constant 54
- kCTFontTableOpbd constant 54
- kCTFontTableOptionExcludeSynthetic constant 55
- kCTFontTableOptionNoOptions constant 55
- kCTFontTableOS2 constant 51
- kCTFontTablePCLT constant 51
- kCTFontTablePost constant 54
- kCTFontTablePrep constant 54
- kCTFontTableProp constant 54
- kCTFontTableTrak constant 55
- kCTFontTableVDMX constant 51
- kCTFontTableVhea constant 55
- kCTFontTableVmtx constant 55
- kCTFontTableVORG constant 51
- kCTFontTableZapf constant 51
- kCTFontToolbarFontType constant 48
- kCTFontToolTipFontType constant 48
- kCTFontTrademarkNameKey constant 42
- kCTFontTraitsAttribute constant 71
- kCTFontTransitionalSerifsClass constant 77
- kCTFontUIOptimizedTrait constant 76
- kCTFontUniqueNameKey constant 41
- kCTFontUnknownClass constant 76
- kCTFontUserFixedPitchFontType constant 46
- kCTFontUserFontType constant 46
- kCTFontUtilityWindowTitleFontType constant 47
- kCTFontVariationAttribute constant 71
- kCTFontVariationAxisDefaultValueKey constant 43
- kCTFontVariationAxisIdentifierKey constant 43
- kCTFontVariationAxisMaximumValueKey constant 43
- kCTFontVariationAxisMinimumValueKey constant 43
- kCTFontVariationAxisNameKey constant 43
- kCTFontVendorURLNameKey constant 42
- kCTFontVersionNameKey constant 41
- kCTFontVerticalOrientation constant 73
- kCTFontVerticalTrait constant 76
- kCTFontViewsFontType constant 46
- kCTFontWeightTrait constant 74
- kCTFontWidthTrait constant 74
- kCTFontWindowTitleFontType constant 47
- kCTForegroundColorAttributeName constant 142
- kCTFrameProgressionAttributeName constant 84
- kCTFrameProgressionAttributeName constant 84
- kCTFrameProgressionRightToLeft constant 84
- kCTFrameProgressionTopToBottom constant 84
- kCTGlyphInfoAttributeName constant 142
- kCTIdentityMappingCharacterCollection constant 93
- kCTJustifiedTextAlignment constant 109
- kCTKernAttributeName constant 141
- kCTLeftTextAlignment constant 109
- kCTLigatureAttributeName constant 142
- kCTLineBreakByCharWrapping constant 110
- kCTLineBreakByClipping constant 110
- kCTLineBreakByTruncatingHead constant 110
- kCTLineBreakByTruncatingMiddle constant 110
- kCTLineBreakByTruncatingTail constant 110
- kCTLineBreakByWordWrapping constant 110
- kCTLineTruncationEnd constant 103
- kCTLineTruncationMiddle constant 104
- kCTLineTruncationStart constant 103
- kCTNaturalTextAlignment constant 109
- kCTParagraphStyleAttributeName constant 142
- kCTParagraphStyleSpecifierAlignment constant 112
- kCTParagraphStyleSpecifierBaseWritingDirection constant 114
- kCTParagraphStyleSpecifierCount constant 114
- kCTParagraphStyleSpecifierDefaultTabInterval constant 113

kCTParagraphStyleSpecifierFirstLineHeadIndent  
     **constant** [112](#)  
 kCTParagraphStyleSpecifierHeadIndent **constant**  
     [112](#)  
 kCTParagraphStyleSpecifierLineBreakMode  
     **constant** [113](#)  
 kCTParagraphStyleSpecifierLineHeightMultiple  
     **constant** [113](#)  
 kCTParagraphStyleSpecifierLineSpacing **constant**  
     [113](#)  
 kCTParagraphStyleSpecifierMaximumLineHeight  
     **constant** [113](#)  
 kCTParagraphStyleSpecifierMinimumLineHeight  
     **constant** [113](#)  
 kCTParagraphStyleSpecifierParagraphSpacing  
     **constant** [113](#)  
 kCTParagraphStyleSpecifierParagraphSpacingBefore  
     **constant** [114](#)  
 kCTParagraphStyleSpecifierTabStops **constant**  
     [112](#)  
 kCTParagraphStyleSpecifierTailIndent **constant**  
     [112](#)  
 kCTRightTextAlignment **constant** [109](#)  
 kCTRunStatusHasNonIdentityMatrix **constant** [124](#)  
 kCTRunStatusNonMonotonic **constant** [124](#)  
 kCTRunStatusNoStatus **constant** [123](#)  
 kCTRunStatusRightToLeft **constant** [124](#)  
 kCTTabColumnTerminatorsAttributeName [128](#)  
 kCTTabColumnTerminatorsAttributeName **constant**  
     [128](#)  
 kCTTypesetterOptionDisableBidiProcessing [133](#)  
 kCTTypesetterOptionDisableBidiProcessing  
     **constant** [133](#)  
 kCTUnderlinePatternDash **constant** [144](#)  
 kCTUnderlinePatternDashDot **constant** [144](#)  
 kCTUnderlinePatternDashDotDot **constant** [144](#)  
 kCTUnderlinePatternDot **constant** [144](#)  
 kCTUnderlinePatternSolid **constant** [144](#)  
 kCTUnderlineStyleAttributeName **constant** [142](#)  
 kCTUnderlineStyleDouble **constant** [143](#)  
 kCTUnderlineStyleNone **constant** [143](#)  
 kCTUnderlineStyleSingle **constant** [143](#)  
 kCTUnderlineStyleThick **constant** [143](#)  
 kCTVersionNumber10\_5 **constant** [138](#)  
 kCTVerticalFormsAttributeName **constant** [142](#)  
 kCTWritingDirectionLeftToRight **constant** [111](#)  
 kCTWritingDirectionNatural **constant** [111](#)  
 kCTWritingDirectionRightToLeft **constant** [111](#)

## N

---

Name Specifier Constants [40](#)

## S

---

String Attribute Name Constants [141](#)

## U

---

User Interface Type Constants [45](#)