# CTParagraphStyle Reference

**Carbon > Text & Fonts**

**2007-04-21**

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



Simultaneously published in the United States and Canada.

# Contents

# CTParagraphStyle Reference

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

## Overview

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

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

## Functions by Task

### Creating Paragraph Styles

`CTParagraphStyleCreate` (page 6)
Creates an immutable paragraph style.

`CTParagraphStyleCreateCopy` (page 6)
Creates an immutable copy of a paragraph style.

### Getting the Value of a Style Specifier

`CTParagraphStyleGetValueForSpecifier` (page 7)
Obtains the current value for a single setting specifier.

### Getting the Type Identifier

`CTParagraphStyleGetTypeID` (page 7)
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.

```
CFTypeID 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 11) 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 11) 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 8). Default: `kCTNaturalTextAlignment` (page 9). 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`. 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 9). Default: `kCTLineBreakByWordWrapping` (page 10). 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 10). Default: `kCTWritingDirectionNatural` (page 11). 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`

# Document Revision History

This table describes the changes to *CTParagraphStyle Reference*.

| Date | Notes |
| --- | --- |
| 2007-04-21 | New document that describes the Core Text opaque type used to represent paragraph or ruler attributes in an attributed string. |

Document Revision History

# Index

## C



## K