CFStringTokenizer Reference

Core Foundation > Internationalization

2009-01-06

Contents

CFStringTokenizer Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Companion guide	Strings Programming Guide for Core Foundation
Declared in	CFStringTokenizer.h

Overview

CFStringTokenizer allows you tokenize strings into words, sentences or paragraphs in a language-neutral way. It supports languages such as Japanese and Chinese that do not delimit words by spaces, as well as de-compounding German compounds. You can obtain Latin transcription for tokens. It also provides language identification API.

You can use a CFStringTokenizer to break a string into tokens (sub-strings) on the basis of words, sentences, or paragraphs. When you create a tokenizer, you can supply options to further modify the tokenization—see "Tokenization Modifiers" (page 13).

In addition, with CFStringTokenizer:

- You can de-compound German compounds
- You can identify the language used in a string (using `CFStringTokenizerCopyBestStringLanguage` (page 8))
- You can obtain Latin transcription for tokens

To find a token that includes the character specified by character index and set it as the current token, you call `CFStringTokenizerGoToTokenAtIndex` (page 11). To advance to the next token and set it as the current token, you call `CFStringTokenizerAdvanceToNextToken` (page 7). To get the range of current token, you call `CFStringTokenizerGetCurrentTokenRange` (page 11). You can use `CFStringTokenizerCopyCurrentTokenAttribute` (page 8) to get the attribute of the current token. If the current token is a compound, you can call `CFStringTokenizerGetCurrentSubTokens` (page 10) to retrieve the subtokens or derived subtokens contained in the compound token. To guess the language of a string, you call `CFStringTokenizerCopyBestStringLanguage` (page 8).

CFStringTokenizer replaces the Language Analysis Manager (see *Language Analysis Manager Reference*). The Language Analysis Manager API provides access to one specific language engine at a time. For example you can create an analysis environment for Japanese tokenization, but it can't then be used to tokenize Chinese. Such API is good when you develop a language specific applications that handle a specific language such as input methods. It is not, however, convenient when you develop an internationalized applications which handle text in language neutral way. Conceptually, CFStringTokenizer provides a higher level API that supports typical tasks of internationalized applications. With CFStringTokenizer you can tokenize a string without knowing the language.

The following Language Analysis Manager functionality is not available with CFStringTokenizer:

- Obtaining the part of speech for a token
- Obtaining alternative tokenization
- Kana-Kanji conversion

Functions by Task

Creating a Tokenizer

`CFStringTokenizerCreate` (page 9)
Returns a tokenizer for a given string.

Setting the String

`CFStringTokenizerSetString` (page 12)
Sets the string for a tokenizer.

Changing the Location

`CFStringTokenizerAdvanceToNextToken` (page 7)
Advances the tokenizer to the next token and sets that as the current token.

`CFStringTokenizerGoToTokenAtIndex` (page 11)
Finds a token that includes the character at a given index, and set it as the current token.

Getting Information About the Current Token

`CFStringTokenizerCopyCurrentTokenAttribute` (page 8)
Returns a given attribute of the current token.

`CFStringTokenizerGetCurrentTokenRange` (page 11)
Returns the range of the current token.

`CFStringTokenizerGetCurrentSubTokens` (page 10)
Retrieves the subtokens or derived subtokens contained in the compound token.

Identifying a Language

`CFStringTokenizerCopyBestStringLanguage` (page 8)
Guesses a language of a given string and returns the guess as a BCP 47 string.

Getting the CFStringTokenizer Type ID

`CFStringTokenizerGetTypeID` (page 11)
Returns the type ID for CFStringTokenizer.

Functions

CFStringTokenizerAdvanceToNextToken

Advances the tokenizer to the next token and sets that as the current token.

```
CFStringTokenizerTokenType CFStringTokenizerAdvanceToNextToken (
   CFStringTokenizerRef tokenizer
);
```

Parameters

`tokenizer`
A CFStringTokenizer object.

Return Value
The type of the token if the tokenizer succeeded in finding a token and setting it as current token. Returns `kCFStringTokenizerTokenNone` if the tokenizer failed to find a token. For possible values, see "Token Types" (page 15).

Discussion
If there is no preceding call to `CFStringTokenizerGoToTokenAtIndex` (page 11) or `CFStringTokenizerAdvanceToNextToken`, the function finds the first token in the range specified by the `CFStringTokenizerCreate` (page 9). If there is a preceding, successful, call to `CFStringTokenizerGoToTokenAtIndex` (page 11) or `CFStringTokenizerAdvanceToNextToken` and there is a current token, proceeds to the next token. If a token is found, it is set as the current token and the function returns `true`; otherwise the current token is invalidates and the function returns `false`.

You can obtain the range and attribute of the token calling `CFStringTokenizerGetCurrentTokenRange` (page 11) and `CFStringTokenizerCopyCurrentTokenAttribute` (page 8). If the token is a compound (with type `kCFStringTokenizerTokenHasSubTokensMask` or `kCFStringTokenizerTokenHasDerivedSubTokensMask`), you can obtain its subtokens and (or) derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 10).

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

See Also
`CFStringTokenizerGoToTokenAtIndex` (page 11)

Declared In
`CFStringTokenizer.h`

CFStringTokenizerCopyBestStringLanguage

Guesses a language of a given string and returns the guess as a BCP 47 string.

```
CFStringRef CFStringTokenizerCopyBestStringLanguage (
   CFStringRef string,
   CFRange range
);
```

Parameters

`string`
The string to test to identify the language.

`range`
The range of *`string`* to use for the test. If `NULL`, the first few hundred characters of the string are examined.

Return Value
A language in BCP 47 form, or `NULL` if the language in *`string`* could not be identified. Ownership follows the Create Rule.

Discussion
The result is not guaranteed to be accurate. Typically, the function requires 200-400 characters to reliably guess the language of a string.

CRStringTokenizer recognizes the following languages:

ar (Arabic), bg (Bulgarian), cs (Czech), da (Danish), de (German), el (Greek), en (English), es (Spanish), fi (Finnish), fr (French), he (Hebrew), hr (Croatian), hu (Hungarian), is (Icelandic), it (Italian), ja (Japanese), ko (Korean), nb (Norwegian Bokmål), nl (Dutch), pl (Polish), pt (Portuguese), ro (Romanian), ru (Russian), sk (Slovak), sv (Swedish), th (Thai), tr (Turkish), uk (Ukrainian), zh-Hans (Simplified Chinese), zh-Hant (Traditional Chinese).

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerCopyCurrentTokenAttribute

Returns a given attribute of the current token.

```
CFTypeRef CFStringTokenizerCopyCurrentTokenAttribute (
   CFStringTokenizerRef tokenizer,
   CFOptionFlags attribute
);
```

Parameters

`tokenizer`
A CFStringTokenizer object.

`attribute`
: The token attribute to obtain. The value must be `kCFStringTokenizerAttributeLatinTranscription`, or `kCFStringTokenizerAttributeLanguage`.

Return Value
The attribute specified by *`attribute`* of the current token, or `NULL` if the current token does not have the specified attribute or there is no current token. Ownership follows the Create Rule.

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerCreate

Returns a tokenizer for a given string.

```
CFStringTokenizerRef CFStringTokenizerCreate (
   CFAllocatorRef alloc,
   CFStringRef string,
   CFRange range,
   CFOptionFlags options,
   CFLocaleRef locale
);
```

Parameters

`alloc`
: The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

`string`
: The string to tokenize.

`range`
: The range of the characters in *`string`* to tokenize.

`options`
: A tokenization unit option that specifies how *`string`* should be tokenized. The options can be modified by adding unit modifier options to tell the tokenizer to prepare specified attributes when it tokenizes *`string`*.

  For possible values, see "Tokenization Modifiers" (page 13).

`locale`
: A locale that specifies language- or region-specific behavior for the tokenization. You can pass `NULL` to use the default system locale, although this is typically not recommended—instead use `CFLocaleCopyCurrent` to specify the locale of the current user.

  For more information, see "Tokenization Modifiers" (page 13).

Return Value
A tokenizer to analyze the range *`range`* of *`string`* for the given locale and options. Ownership follows the Create Rule.

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerGetCurrentSubTokens

Retrieves the subtokens or derived subtokens contained in the compound token.

```
CFIndex CFStringTokenizerGetCurrentSubTokens (
    CFStringTokenizerRef tokenizer,
    CFRange *ranges,
    CFIndex maxRangeLength,
    CFMutableArrayRef derivedSubTokens
);
```

Parameters

tokenizer

A CFStringTokenizer object.

tokenizer

Upon return, an array of CFRanges containing the ranges of subtokens. The ranges are relative to the string specified to CFStringTokenizerCreate. This parameter can be `NULL`.

maxRangeLength

The maximum number of ranges to return.

derivedSubTokens

A CFMutableArray to which the derived subtokens are to be added. This parameter can be `NULL`.

Return Value
The number of ranges returned.

Discussion
If token type is `kCFStringTokenizerTokenNone`, the *ranges* array and *derivedSubTokens* array are untouched and the return value is `0`.

If token type is `kCFStringTokenizerTokenNormal`, the *ranges* array has one item filled in with the entire range of the token (if `maxRangeLength` >= 1) and a string taken from the entire token range is added to the *derivedSubTokens* array and the return value is `1`.

If token type is `kCFStringTokenizerTokenHasSubTokensMask` or `kCFStringTokenizerTokenHasDerivedSubTokensMask`, the ranges array is filled in with as many items as there are subtokens (up to a limit of *maxRangeLength*).

The *derivedSubTokens* array will have sub tokens added even when the sub token is a substring of the token. If token type is `kCFStringTokenizerTokenHasSubTokensMask`, the ordinary non-derived subtokens are added to the *derivedSubTokens* array.

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerGetCurrentTokenRange

Returns the range of the current token.

```
CFRange CFStringTokenizerGetCurrentTokenRange (
   CFStringTokenizerRef tokenizer
);
```

Parameters

tokenizer

A CFStringTokenizer object.

Return Value

The range of the current token, or `{kCFNotFound,0}` if there is no current token.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStringTokenizer.h`

CFStringTokenizerGetTypeID

Returns the type ID for CFStringTokenizer.

```
CFTypeID CFStringTokenizerGetTypeID (
   void
);
```

Return Value

The type ID for CFStringTokenizer.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStringTokenizer.h`

CFStringTokenizerGoToTokenAtIndex

Finds a token that includes the character at a given index, and set it as the current token.

```
CFStringTokenizerTokenType CFStringTokenizerGoToTokenAtIndex (
   CFStringTokenizerRef tokenizer,
   CFIndex index
);
```

Parameters

tokenizer

A CFStringTokenizer object.

index

The index of a character in the string for *tokenizer*.

Return Value

The type of the token if the tokenizer succeeded in finding a token and setting it as the current token. Returns `kCFStringTokenizerTokenNone` if the tokenizer failed to find a token. For possible values, see "Token Types" (page 15).

Discussion

You can obtain the range and attribute of the token calling `CFStringTokenizerGetCurrentTokenRange` (page 11) and `CFStringTokenizerCopyCurrentTokenAttribute` (page 8). If the token is a compound (with type `kCFStringTokenizerTokenHasSubTokensMask` or `kCFStringTokenizerTokenHasDerivedSubTokensMask`), you can obtain its subtokens and (or) derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 10).

Availability

Available in Mac OS X v10.5 and later.

See Also

`CFStringTokenizerAdvanceToNextToken` (page 7)

Declared In

`CFStringTokenizer.h`

CFStringTokenizerSetString

Sets the string for a tokenizer.

```
void CFStringTokenizerSetString (
    CFStringTokenizerRef tokenizer,
    CFStringRef string,
    CFRange range
);
```

Parameters

`tokenizer`

A tokenizer.

`string`

The string for the tokenizer to tokenize.

`range`

The range of string to tokenize. The range of characters within the string to be tokenized. The specified range must not exceed the length of the string.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStringTokenizer.h`

Data Types

CFStringTokenizerRef

A reference to a CFStringTokenizer object.

```
typedef struct __CFStringTokenizer * CFStringTokenizerRef;
```

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerTokenType

Token types returned by `CFStringTokenizerGoToTokenAtIndex` (page 11) and `CFStringTokenizerAdvanceToNextToken` (page 7).

```
typedef CFOptionFlags CFStringTokenizerTokenType;
```

Discussion
For possible values, see "Token Types" (page 15).

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

Declared In
`CFStringTokenizer.h`

Constants

Tokenization Modifiers

Tokenization options are used with `CFStringTokenizerCreate` (page 9) to specify how the string should be tokenized

```
enum {
    kCFStringTokenizerUnitWord      = 0,
    kCFStringTokenizerUnitSentence  = 1,
    kCFStringTokenizerUnitParagraph = 2,
    kCFStringTokenizerUnitLineBreak = 3,
    kCFStringTokenizerUnitWordBoundary = 4,
    kCFStringTokenizerAttributeLatinTranscription = 1L << 16,
    kCFStringTokenizerAttributeLanguage           = 1L << 17
};
```

Constants

`kCFStringTokenizerUnitWord`

Specifies that a string should be tokenized by word. The *`locale`* parameter of `CFStringTokenizerCreate` (page 9) is ignored.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitSentence`

Specifies that a string should be tokenized by sentence. The *`locale`* parameter of `CFStringTokenizerCreate` (page 9) is ignored.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitParagraph`

Specifies that a string should be tokenized by paragraph. The *`locale`* parameter of `CFStringTokenizerCreate` (page 9) is ignored.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitLineBreak`

Specifies that a string should be tokenized by line break. The *`locale`* parameter of `CFStringTokenizerCreate` (page 9) is ignored.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitWordBoundary`

Specifies that a string should be tokenized by locale-sensitive word boundary.

You can use this constant in double-click range detection and whole word search. It is locale-sensitive. If the locale is `en_US_POSIX`, a colon (U+003A) is treated as a word separator. If the *`locale`* parameter of `CFStringTokenizerCreate` (page 9) is `NULL`, the locale from the global `AppleTextBreakLocale` preference is used if it is available; otherwise the locale defaults to the first locale in `AppleLanguages`.

`kCFStringTokenizerUnitWordBoundary` also returns space between words as a token.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerAttributeLatinTranscription`

Used with `kCFStringTokenizerUnitWord`, tells the tokenizer to prepare the Latin transcription when it tokenizes the string.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerAttributeLanguage`
Tells the tokenizer to prepare the language (specified as an RFC 3066bis string) when it tokenizes the string.

Used with `kCFStringTokenizerUnitSentence` (page 14) or `kCFStringTokenizerUnitParagraph` (page 14).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

Discussion
You use the tokenization unit options with `CFStringTokenizerCreate` (page 9) to specify how a string should be tokenized.

You use the modifiers together with a tokenization unit to modify the way the string is tokenized.

You use the attribute specifiers to tell the tokenizer to prepare the specified attribute when it tokenizes the given string. You can retrieve the attribute value by calling `CFStringTokenizerCopyCurrentTokenAttribute` (page 8) with one of the attribute options.

The locale sensitivity of the tokenization unit options may change in a future release.

Declared In
`CFStringTokenizer.h`

Token Types

Token types returned by `CFStringTokenizerGoToTokenAtIndex` (page 11) and `CFStringTokenizerAdvanceToNextToken` (page 7).

```
enum {
    kCFStringTokenizerTokenNone                    = 0,
    kCFStringTokenizerTokenNormal                  = 1,
    kCFStringTokenizerTokenHasSubTokensMask        = 1L << 1,
    kCFStringTokenizerTokenHasDerivedSubTokensMask = 1L << 2,
    kCFStringTokenizerTokenHasHasNumbersMask       = 1L << 3,
    kCFStringTokenizerTokenHasNonLettersMask       = 1L << 4,
    kCFStringTokenizerTokenIsCJWordMask            = 1L << 5
};
```

Constants
`kCFStringTokenizerTokenNone`
Has no token.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenNormal`
Has a normal token.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasSubTokensMask`
Compound token which may contain subtokens but with no derived subtokens.

You can obtain subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 10).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasDerivedSubTokensMask`
Compound token which may contain derived subtokens.

You can obtain subtokens and derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 10).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasHasNumbersMask`
Appears to contain a number.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasNonLettersMask`
Contains punctuation, symbols, and so on.

Given the way Unicode word break works, this means it is a standalone punctuation or symbol character, or a string of such.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenIsCJWordMask`
Contains kana and/or ideographs.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

Discussion
See http://www.unicode.org/reports/tr29/#Word_Boundaries for a detailed description of word boundaries.

Declared In
`CFStringTokenizer.h`

Document Revision History

This table describes the changes to *CFStringTokenizer Reference*.

Date	Notes
2009-01-06	Added information about the locale sensitivity of each token unit used with the CFStringTokenizerCreate function.
2007-10-31	Added new API.
2007-05-21	New document that describes the opaque typed used to break a string into tokens (sub-strings) on the basis of words, sentences, or paragraphs.

Index