CFString Reference

Core Foundation

2008-10-15

Contents

CFString Reference 7

Overview 7
Functions by Task 8
- Creating a CFString 8
- Searching Strings 9
- Comparing Strings 9
- Accessing Characters 9
- Working With Encodings 10
- Getting Numeric Values 11
- Getting String Properties 11
- String File System Representations 11
- Getting Paragraph Bounds 12

Functions 12
- CFShowStr 12
- CFSTR 13
- CFStringCompare 13
- CFStringCompareWithOptions 14
- CFStringCompareWithOptionsAndLocale 15
- CFStringConvertEncodingToIANACharSetName 16
- CFStringConvertEncodingToNSStringEncoding 16
- CFStringConvertEncodingToWindowsCodepage 17
- CFStringConvertIANACharSetNameToEncoding 17
- CFStringConvertNSStringEncodingToEncoding 18
- CFStringConvertWindowsCodepageToEncoding 18
- CFStringCreateArrayBySeparatingStrings 19
- CFStringCreateArrayWithFindResults 20
- CFStringCreateByCombiningStrings 21
- CFStringCreateCopy 21
- CFStringCreateExternalRepresentation 22
- CFStringCreateFromExternalRepresentation 23
- CFStringCreateWithBytes 24
- CFStringCreateWithBytesNoCopy 25
- CFStringCreateWithCharacters 26
- CFStringCreateWithCharactersNoCopy 27
- CFStringCreateWithCString 28
- CFStringCreateWithCStringNoCopy 29
- CFStringCreateWithFileSystemRepresentation 30
- CFStringCreateWithFormat 31
- CFStringCreateWithFormatAndArguments 32
- CFStringCreateWithPascalString 32
- CFStringCreateWithPascalStringNoCopy 33

CFStringCreateWithSubstring 34
CFStringFind 35
CFStringFindCharacterFromSet 36
CFStringFindWithOptions 37
CFStringFindWithOptionsAndLocale 38
CFStringGetBytes 39
CFStringGetCharacterAtIndex 40
CFStringGetCharacterFromInlineBuffer 41
CFStringGetCharacters 41
CFStringGetCharactersPtr 42
CFStringGetCString 43
CFStringGetCStringPtr 44
CFStringGetDoubleValue 44
CFStringGetFastestEncoding 45
CFStringGetFileSystemRepresentation 45
CFStringGetIntValue 46
CFStringGetLength 47
CFStringGetLineBounds 47
CFStringGetListOfAvailableEncodings 48
CFStringGetMaximumSizeForEncoding 49
CFStringGetMaximumSizeOfFileSystemRepresentation 49
CFStringGetMostCompatibleMacStringEncoding 50
CFStringGetNameOfEncoding 50
CFStringGetParagraphBounds 51
CFStringGetPascalString 52
CFStringGetPascalStringPtr 53
CFStringGetRangeOfComposedCharactersAtIndex 53
CFStringGetSmallestEncoding 54
CFStringGetSystemEncoding 54
CFStringGetTypeID 55
CFStringHasPrefix 56
CFStringHasSuffix 56
CFStringInitInlineBuffer 57
CFStringIsEncodingAvailable 57
Data Types 58
CFStringCompareFlags 58
CFStringEncoding 58
CFStringEncodings 59
CFStringInlineBuffer 59
CFStringRef 59
Constants 60
String Comparison Flags 60
Built-in String Encodings 61
Invalid String Encoding Flag 63
External String Encodings 64

Document Revision History 79

Index 81

CFString Reference

Derived From:	CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBase.h CFString.h CFStringEncodingExt.h
Companion guides	Property List Programming Topics for Core Foundation Strings Programming Guide for Core Foundation Data Formatting Guide for Core Foundation

Overview

CFString provides a suite of efficient string-manipulation and string-conversion functions. It offers seamless Unicode support and facilitates the sharing of data between Carbon and Cocoa programs. CFString is relevant for any Carbon application that uses strings. If your application supports (or is planning to support) Unicode, CFString is recommended. CFString creates immutable strings—use CFMutableString to create and manage a string that can be changed after it has been created.

CFString has two primitive functions, `CFStringGetLength` (page 47) and `CFStringGetCharacterAtIndex` (page 40), that provide the basis for all other functions in its interface. The `CFStringGetLength` function returns the total number (in terms of UTF-16 code pairs) of characters in the string. The `CFStringGetCharacterAtIndex` function gives access to each character in the string by index, with index values starting at `0`.

CFString provides functions for finding and comparing strings. It also provides functions for reading numeric values from strings, for combining strings in various ways, and for converting a string to different forms (such as encoding and case changes). A number of functions, for example `CFStringFindWithOptions`, allow you to specify a range over which to operate within a string. The specified range must not exceed the length of the string. Debugging options may help you to catch any errors that arise if a range does exceed a string's length.

Like other Core Foundation types, CFStrings can be hashed using the `CFHash` function. Note, though, that hash values are not guaranteed to remain equal between releases of the operating system. In particular, hash values are different between Mac OS X v10.3 and v10.4. If you need to make a hash value persistent and consistent across different releases, you should use an alternate technique, such as SHA-1.

CFString is "toll-free bridged" with its Cocoa Foundation counterpart, NSString. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSString *` parameter, you can pass in a `CFStringRef`, and in a function where you see a `CFStringRef` parameter, you can pass in an NSString instance. This also applies to concrete subclasses of NSString. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a CFString

`CFSTR` (page 13)
Creates an immutable string from a constant compile-time string.

`CFStringCreateArrayBySeparatingStrings` (page 19)
Creates an array of CFString objects from a single CFString object.

`CFStringCreateByCombiningStrings` (page 21)
Creates a single string from the individual CFString objects that comprise the elements of an array.

`CFStringCreateCopy` (page 21)
Creates an immutable copy of a string.

`CFStringCreateFromExternalRepresentation` (page 23)
Creates a string from its "external representation."

`CFStringCreateWithBytes` (page 24)
Creates a string from a buffer containing characters in a specified encoding.

`CFStringCreateWithBytesNoCopy` (page 25)
Creates a string from a buffer, containing characters in a specified encoding, that might serve as the backing store for the new string.

`CFStringCreateWithCharacters` (page 26)
Creates a string from a buffer of Unicode characters.

`CFStringCreateWithCharactersNoCopy` (page 27)
Creates a string from a buffer of Unicode characters that might serve as the backing store for the object.

`CFStringCreateWithCString` (page 28)
Creates an immutable string from a C string.

`CFStringCreateWithCStringNoCopy` (page 29)
Creates a CFString object from an external C string buffer that might serve as the backing store for the object.

`CFStringCreateWithFormat` (page 31)
Creates an immutable string from a formatted string and a variable number of arguments.

`CFStringCreateWithFormatAndArguments` (page 32)
Creates an immutable string from a formatted string and a variable number of arguments (specified in a parameter of type `va_list`).

`CFStringCreateWithPascalString` (page 32)
Creates an immutable CFString object from a Pascal string.

`CFStringCreateWithPascalStringNoCopy` (page 33)
Creates a CFString object from an external Pascal string buffer that might serve as the backing store for the object.

`CFStringCreateWithSubstring` (page 34)
Creates an immutable string from a segment (substring) of an existing string.

Searching Strings

`CFStringCreateArrayWithFindResults` (page 20)
: Searches a string for multiple occurrences of a substring and creates an array of ranges identifying the locations of these substrings within the target string.

`CFStringFind` (page 35)
: Searches for a substring within a string and, if it is found, yields the range of the substring within the object's characters.

`CFStringFindCharacterFromSet` (page 36)
: Query the range of the first character contained in the specified character set.

`CFStringFindWithOptions` (page 37)
: Searches for a substring within a range of the characters represented by a string and, if the substring is found, returns its range within the object's characters.

`CFStringFindWithOptionsAndLocale` (page 38)
: Returns a Boolean value that indicates whether a given string was found in a given source string.

`CFStringGetLineBounds` (page 47)
: Given a range of characters in a string, obtains the line bounds—that is, the indexes of the first character and the final characters of the lines containing the range.

Comparing Strings

`CFStringCompare` (page 13)
: Compares one string with another string.

`CFStringCompareWithOptions` (page 14)
: Compares a range of the characters in one string with that of another string.

`CFStringCompareWithOptionsAndLocale` (page 15)
: Compares a range of the characters in one string with another string using a given locale.

`CFStringHasPrefix` (page 56)
: Determines if the character data of a string begin with a specified sequence of characters.

`CFStringHasSuffix` (page 56)
: Determines if a string ends with a specified sequence of characters.

Accessing Characters

`CFStringCreateExternalRepresentation` (page 22)
: Creates an “external representation” of a CFString object, that is, a CFData object.

`CFStringGetBytes` (page 39)
: Fetches a range of the characters from a string into a byte buffer after converting the characters to a specified encoding.

`CFStringGetCharacterAtIndex` (page 40)
: Returns the Unicode character at a specified location in a string.

`CFStringGetCharacters` (page 41)
: Copies a range of the Unicode characters from a string to a user-provided buffer.

`CFStringGetCharactersPtr` (page 42)
: Quickly obtains a pointer to the contents of a string as a buffer of Unicode characters.

`CFStringGetCharacterFromInlineBuffer` (page 41)
: Returns the Unicode character at a specific location in an in-line buffer.

`CFStringGetCString` (page 43)
: Copies the character contents of a string to a local C string buffer after converting the characters to a given encoding.

`CFStringGetCStringPtr` (page 44)
: Quickly obtains a pointer to a C-string buffer containing the characters of a string in a given encoding.

`CFStringGetLength` (page 47)
: Returns the number (in terms of UTF-16 code pairs) of Unicode characters in a string.

`CFStringGetPascalString` (page 52)
: Copies the character contents of a CFString object to a local Pascal string buffer after converting the characters to a requested encoding.

`CFStringGetPascalStringPtr` (page 53)
: Quickly obtains a pointer to a Pascal buffer containing the characters of a string in a given encoding.

`CFStringGetRangeOfComposedCharactersAtIndex` (page 53)
: Returns the range of the composed character sequence at a specified index.

`CFStringInitInlineBuffer` (page 57)
: Initializes an in-line buffer to use for efficient access of a CFString object's characters.

Working With Encodings

`CFStringConvertEncodingToIANACharSetName` (page 16)
: Returns the name of the IANA registry “charset” that is the closest mapping to a specified string encoding.

`CFStringConvertEncodingToNSStringEncoding` (page 16)
: Returns the Cocoa encoding constant that maps most closely to a given Core Foundation encoding constant.

`CFStringConvertEncodingToWindowsCodepage` (page 17)
: Returns the Windows codepage identifier that maps most closely to a given Core Foundation encoding constant.

`CFStringConvertIANACharSetNameToEncoding` (page 17)
: Returns the Core Foundation encoding constant that is the closest mapping to a given IANA registry “charset” name.

`CFStringConvertNSStringEncodingToEncoding` (page 18)
: Returns the Core Foundation encoding constant that is the closest mapping to a given Cocoa encoding.

`CFStringConvertWindowsCodepageToEncoding` (page 18)
: Returns the Core Foundation encoding constant that is the closest mapping to a given Windows codepage identifier.

`CFStringGetFastestEncoding` (page 45)
: Returns for a CFString object the character encoding that requires the least conversion time.

`CFStringGetListOfAvailableEncodings` (page 48)
: Returns a pointer to a list of string encodings supported by the current system.

`CFStringGetMaximumSizeForEncoding` (page 49)
: Returns the maximum number of bytes a string of a specified length (in Unicode characters) will take up if encoded in a specified encoding.

`CFStringGetMostCompatibleMacStringEncoding` (page 50)
: Returns the most compatible Mac OS script value for the given input encoding.

`CFStringGetNameOfEncoding` (page 50)
: Returns the canonical name of a specified string encoding.

`CFStringGetSmallestEncoding` (page 54)
: Returns the smallest encoding on the current system for the character contents of a string.

`CFStringGetSystemEncoding` (page 54)
: Returns the default encoding used by the operating system when it creates strings.

`CFStringIsEncodingAvailable` (page 57)
: Determines whether a given Core Foundation string encoding is available on the current system.

Getting Numeric Values

`CFStringGetDoubleValue` (page 44)
: Returns the primary `double` value represented by a string.

`CFStringGetIntValue` (page 46)
: Returns the integer value represented by a string.

Getting String Properties

`CFShowStr` (page 12)
: Prints the attributes of a string during debugging.

`CFStringGetTypeID` (page 55)
: Returns the type identifier for the CFString opaque type.

String File System Representations

`CFStringCreateWithFileSystemRepresentation` (page 30)
: Creates a CFString from a zero-terminated POSIX file system representation.

`CFStringGetFileSystemRepresentation` (page 45)
: Extracts the contents of a string as a `NULL`-terminated 8-bit string appropriate for passing to POSIX APIs.

`CFStringGetMaximumSizeOfFileSystemRepresentation` (page 49)
: Determines the upper bound on the number of bytes required to hold the file system representation of the string.

Getting Paragraph Bounds

`CFStringGetParagraphBounds` (page 51)
: Given a range of characters in a string, obtains the paragraph bounds—that is, the indexes of the first character and the final characters of the paragraph(s) containing the range.

Functions

CFShowStr

Prints the attributes of a string during debugging.

```
void CFShowStr (
   CFStringRef str
);
```

Parameters

`str`
: The string whose attributes you want to print.

Discussion

Use this function to learn about specific attributes of a CFString object during debugging. These attributes include the following:

- Length (in Unicode characters)
- Whether originally it was an 8-bit string and, if so, whether it was a C (`HasNullByte`) or Pascal (`HasLengthByte`) string
- Whether it is a mutable or an immutable object
- The allocator used to create it
- The memory address of the character contents and whether those contents are in-line

The information provided by this function is for debugging purposes only. The values of any of these attributes might change between different releases and on different platforms. Note in particular that this function does not show the contents of the string. If you want to display the contents of the string, use `CFShow`.

Special Considerations

You can use `CFShowStr` in one of two general ways. If your debugger supports function calls (such as `gdb` does), call `CFShowStr` in the debugger:

```
(gdb) call (void) CFShowStr(string)
Length 11
IsEightBit 1
HasLengthByte 1
HasNullByte 1
InlineContents 1
Allocator SystemDefault
Mutable 0
Contents 0x4e7c0
```

You can also incorporate calls to `CFShowStr` in a test version of your code to print descriptions of CFString objects to the console.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFSTR

Creates an immutable string from a constant compile-time string.

```
CFStringRef CFSTR (
    const char *cStr
);
```

Parameters

`cStr`
A constant C string (that is, text enclosed in double-quotation marks) from which the string is to be created. The characters enclosed by the quotation marks must be ASCII characters, otherwise the behavior is undefined.

Return Value
An immutable string, or `NULL` if there was a problem creating the object. The returned object is a constant. You may retain and release it, similar to other immutable CFString objects, but are not required to do so—it will remain valid until the program terminates.

Discussion
The `CFSTR` macro is a convenient way to create CFString representations of constant compile-time strings.

A value returned by `CFSTR` has the following semantics:

- Values returned from `CFSTR` are not released by CFString—they are guaranteed to be valid until the program terminates.
- Values returned from `CFSTR` can be retained and released in a balanced fashion, like any other CFString, but you are not required to do so.

Non-ASCII characters (that is, character codes greater than 127) are not supported. If you use them, the result is undefined. Even if using them works for you in testing, it might not work if the user selects a different language preference.

Note that when using this macro as an initializer, you must compile using the flag `-fconstant-cfstrings` (see Options Controlling C Dialect).

CFStringCompare

Compares one string with another string.

```
CFComparisonResult CFStringCompare (
   CFStringRef theString1,
   CFStringRef theString2,
   CFOptionFlags compareOptions
);
```

Parameters

theString1

The first string to use in the comparison.

theString2

The second string to use in the comparison.

compareOptions

Flags that select different types of comparisons, such as localized comparison, case-insensitive comparison, and non-literal comparison. If you want the default comparison behavior, pass `0`. See "String Comparison Flags" (page 60) for the available flags.

Return Value

A `CFComparisonResult` value that indicates whether *theString1* is equal to, less than, or greater than *theString2*.

Discussion

You can affect how the comparison proceeds by specifying one or more option flags in *compareOptions*. Not all comparison options are currently implemented.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImageBrowserView

ImageClient

MoreIsBetter

MoreOSL

NSLMiniBrowser

Declared In

`CFString.h`

CFStringCompareWithOptions

Compares a range of the characters in one string with that of another string.

```
CFComparisonResult CFStringCompareWithOptions (
   CFStringRef theString1,
   CFStringRef theString2,
   CFRange rangeToCompare,
   CFOptionFlags compareOptions
);
```

Parameters

theString1

The first string to use in the comparison.

theString2

The second string to use in the comparison.

rangeToCompare

The range of characters in *theString1* to be used in the comparison to *theString2*. To use the whole string, pass the range `CFRangeMake(0, CFStringGetLength(theString1))` or use `CFStringCompare` (page 13). The specified range must not exceed the length of the string.

compareOptions

Flags that select different types of comparisons, such as localized comparison, case-insensitive comparison, and non-literal comparison. If you want the default comparison behavior, pass `0`. See "String Comparison Flags" (page 60) for the available flags.

Return Value

A `CFComparisonResult` value that indicates whether *theString1* is equal to, less than, or greater than *theString2*.

Discussion

You can affect how the comparison proceeds by specifying one or more option flags in *compareOptions*.

If you want to compare one entire string with another string, use the `CFStringCompare` (page 13) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringCompareWithOptionsAndLocale

Compares a range of the characters in one string with another string using a given locale.

```
CFComparisonResult CFStringCompareWithOptionsAndLocale (
   CFStringRef theString1,
   CFStringRef theString2,
   CFRange rangeToCompare,
   CFOptionFlags compareOptions,
   CFLocaleRef locale
);
```

Parameters

theString1

The first string to use in the comparison.

theString2

The second string to use in the comparison. The full range of this string is used.

rangeToCompare

The range of characters in *theString1* to be used in the comparison to *theString2*. To use the whole string, pass the range `CFRangeMake(0, CFStringGetLength(theString1))`. The specified range must not exceed the bounds of the string.

compareOptions

Flags that select different types of comparisons, such as case-insensitive comparison and non-literal comparison. If you want the default comparison behavior, pass `0`. See "String Comparison Flags" (page 60) for the available flags. `kCFCompareBackwards` and `kCFCompareAnchored` are not applicable.

locale

The locale to use for the comparison. `NULL` specifies the canonical locale (the return value from `CFLocaleGetSystem`). The locale argument affects both equality and ordering algorithms. For example, in some locales, accented characters are ordered immediately after the base; other locales order them after "z".

Return Value

A `CFComparisonResult` value that indicates whether *theString1* is equal to, less than, or greater than *theString2*.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFString.h`

CFStringConvertEncodingToIANACharSetName

Returns the name of the IANA registry "charset" that is the closest mapping to a specified string encoding.

```
CFStringRef CFStringConvertEncodingToIANACharSetName (
   CFStringEncoding encoding
);
```

Parameters

encoding

The Core Foundation string encoding to use.

Return Value

The name of the IANA "charset" that is the closest mapping to *encoding*. Returns `NULL` if the encoding is not recognized.

Discussion

The `CFStringConvertIANACharSetNameToEncoding` (page 17) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringConvertEncodingToNSStringEncoding

Returns the Cocoa encoding constant that maps most closely to a given Core Foundation encoding constant.

```
UInt32 CFStringConvertEncodingToNSStringEncoding (
   CFStringEncoding encoding
);
```

Parameters

encoding

The Core Foundation string encoding to use.

Return Value

The Cocoa encoding (of type `NSStringEncoding`) that is closest to the Core Foundation encoding *`encoding`*. The behavior is undefined if an invalid string encoding is passed.

Discussion

The `CFStringConvertNSStringEncodingToEncoding` (page 18) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`CFString.h`

CFStringConvertEncodingToWindowsCodepage

Returns the Windows codepage identifier that maps most closely to a given Core Foundation encoding constant.

```
UInt32 CFStringConvertEncodingToWindowsCodepage (
   CFStringEncoding encoding
);
```

Parameters

`encoding`

The Core Foundation string encoding to use.

Return Value

The Windows codepage value that is closest to the Core Foundation encoding *`encoding`*. The behavior is undefined if an invalid string encoding is passed.

Discussion

The `CFStringConvertWindowsCodepageToEncoding` (page 18) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringConvertIANACharSetNameToEncoding

Returns the Core Foundation encoding constant that is the closest mapping to a given IANA registry "charset" name.

```
CFStringEncoding CFStringConvertIANACharSetNameToEncoding (
   CFStringRef theString
);
```

Parameters

IANAName
: The IANA “charset” name to use.

Return Value

The Core Foundation string encoding that is closest to the IANA “charset” *IANAName*. Returns the `kCFStringEncodingInvalidId` (page 64) constant if the name is not recognized.

Discussion

The `CFStringConvertEncodingToIANACharSetName` (page 16) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringConvertNSStringEncodingToEncoding

Returns the Core Foundation encoding constant that is the closest mapping to a given Cocoa encoding.

```
CFStringEncoding CFStringConvertNSStringEncodingToEncoding (
   UInt32 encoding
);
```

Parameters

encoding
: The Cocoa string encoding (of type `NSStringEncoding`) to use.

Return Value

The Core Foundation string encoding that is closest to the Cocoa string encoding *encoding*. Returns the `kCFStringEncodingInvalidId` (page 64) constant if the mapping is not known.

Discussion

The `CFStringConvertEncodingToNSStringEncoding` (page 16) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringConvertWindowsCodepageToEncoding

Returns the Core Foundation encoding constant that is the closest mapping to a given Windows codepage identifier.

```
CFStringEncoding CFStringConvertWindowsCodepageToEncoding (
   UInt32 codepage
);
```

Parameters

codepage

The Windows codepage identifier to use.

Return Value

The Core Foundation string encoding that is closest to the Windows codepage identifier *codepage*. Returns the `kCFStringEncodingInvalidId` (page 64) constant if the mapping is not known.

Discussion

The `CFStringConvertEncodingToWindowsCodepage` (page 17) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringCreateArrayBySeparatingStrings

Creates an array of CFString objects from a single CFString object.

```
CFArrayRef CFStringCreateArrayBySeparatingStrings (
   CFAllocatorRef alloc,
   CFStringRef theString,
   CFStringRef separatorString
);
```

Parameters

alloc

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

theString

The string to be divided into substrings. The substrings should be separated by *separatorString*.

separatorString

A string containing the character or characters used to separate the substrings in *theString*.

Return Value

A new array that contains CFString objects that represent substrings of *theString*, or `NULL` if there was a problem creating the object. The order of elements in the array is identical to the order of the substrings in *theString*. If *separatorString* does not occur in *theString*, the result is an array containing *theString*. If *separatorString* is equal to *theString*, then the result is an array containing two empty strings. Ownership follows the Create Rule.

Discussion

This function provides a convenient way to convert units of data captured in a single string to a form (an array) suitable for iterative processing. One or more delimiter characters (or "separator string") separates the substrings in the source string—these characters are frequently whitespace characters such as tabs and newlines (carriage returns). For example, you might have a file containing a localized list of place names with each name separated by a tab character. You could create a CFString object from this file and call this function on the string to obtain a CFArray object whose elements are these place names.

See also `CFStringCreateByCombiningStrings` (page 21).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
iTunesController

MoreIsBetter

MoreSCF

QISA

Declared In
`CFString.h`

CFStringCreateArrayWithFindResults

Searches a string for multiple occurrences of a substring and creates an array of ranges identifying the locations of these substrings within the target string.

```
CFArrayRef CFStringCreateArrayWithFindResults (
   CFAllocatorRef alloc,
   CFStringRef theString,
   CFStringRef stringToFind,
   CFRange rangeToSearch,
   CFOptionFlags compareOptions
);
```

Parameters

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

theString
The string in which to search for *stringToFind*.

stringToFind
The string to search for in *theString*.

rangeToSearch
The range of characters within *theString* to be searched. The specified range must not exceed the length of the string.

compareOptions
Flags that select different types of comparisons, such as localized comparison, case-insensitive comparison, and non-literal comparison. If you want the default comparison behavior, pass `0`. See "String Comparison Flags" (page 60) for the available flags.

Return Value
An array that contains pointers to `CFRange` structures identifying the character locations of *stringToFind* in *theString*. Returns `NULL`, if no matching substring is found in the source object, or if there was a problem creating the array. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringCreateByCombiningStrings

Creates a single string from the individual CFString objects that comprise the elements of an array.

```
CFStringRef CFStringCreateByCombiningStrings (
   CFAllocatorRef alloc,
   CFArrayRef theArray,
   CFStringRef separatorString
);
```

Parameters

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

`theArray`
: An array of CFString objects to concatenate. This value should not be `NULL`.

`separatorString`
: The string to insert between the substrings in the returned string. This value is commonly a whitespace character such as a tab or a newline (carriage return). If this value is not a valid CFString object, an assertion is raised.

Return Value
A string that contains a concatenation of the strings in *`theArray`* separated by *`separatorString`*. The order of the substrings in the string is identical to the order of the elements in *`theArray`*.

If *`theArray`* is empty, returns an empty CFString object; if *`theArray`* contains one CFString object, that object is returned (without the separator string). Returns `NULL` if there was a problem in creating the string. Ownership follows the Create Rule.

Discussion
See also `CFStringCreateArrayBySeparatingStrings` (page 19).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MyFirstJNIProject

Declared In
`CFString.h`

CFStringCreateCopy

Creates an immutable copy of a string.

```
CFStringRef CFStringCreateCopy (
   CFAllocatorRef alloc,
   CFStringRef theString
);
```

Parameters

alloc

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

theString

The string to copy.

Return Value

An immutable string whose contents are identical to *theString*. Returns `NULL` if there was a problem copying the object. Ownership follows the Create Rule.

Discussion

The resulting object has the same Unicode contents as the original object, but it is always immutable. It might also have different storage characteristics, and hence might reply differently to functions such as `CFStringGetCStringPtr` (page 44). Also, if the specified allocator and the allocator of the original object are the same, and the string is already immutable, this function may simply increment the retention count without making a true copy. However, the resulting object is a true immutable copy, except the operation was a lot more efficient.

You should use this function in situations where a string is or could be mutable, and you need to take a snapshot of its current value. For example, you might decide to pass a copy of a string to a function that stores its current value in a list for later use.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DockBrowser

HID Calibrator

HID Explorer

NSLMiniBrowser

QTMetaData

Declared In

`CFString.h`

CFStringCreateExternalRepresentation

Creates an "external representation" of a CFString object, that is, a CFData object.

```
CFDataRef CFStringCreateExternalRepresentation (
   CFAllocatorRef alloc,
   CFStringRef theString,
   CFStringEncoding encoding,
   UInt8 lossByte
);
```

Parameters

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

`theString`
: The string to convert to an external representation.

`encoding`
: The string encoding to use for the external representation.

`lossByte`
: The character value to assign to characters that cannot be converted to the requested encoding. Pass `0` if you want conversion to stop at the first such error; if this happens, the function returns `NULL`.

Return Value
A CFData object that stores the characters of the CFString object as an "external representation." Returns `NULL` if no loss byte was specified and the function could not convert the characters to the specified encoding. Ownership follows the Create Rule.

Discussion
In the CFData object form, the string can be written to disk as a file or be sent out over a network. If the encoding of the characters in the data object is Unicode, the function may insert a BOM (byte-order marker) to indicate endianness. However, representations created with encoding constants `kCFStringEncodingUTF16BE`, `kCFStringEncodingUTF16LE`, `kCFStringEncodingUTF32BE`, and `kCFStringEncodingUTF32LE` do not include a BOM because the byte order is explicitly indicated by the letters "BE" (big-endian) and "LE" (little-endian).

This function allows the specification of a "loss byte" to represent characters that cannot be converted to the requested encoding.

When you create an external representation from a CFMutableString object, it loses this mutability characteristic when it is converted back to a CFString object.

The `CFStringCreateFromExternalRepresentation` (page 23) function complements this function by creating a CFString object from an "external representation" CFData object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringCreateFromExternalRepresentation

Creates a string from its "external representation."

```
CFStringRef CFStringCreateFromExternalRepresentation (
   CFAllocatorRef alloc,
   CFDataRef data,
   CFStringEncoding encoding
);
```

Parameters

alloc

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

data

The CFData object containing bytes that hold the characters in the specified encoding.

encoding

The encoding to use when interpreting the bytes in the data argument.

Return Value

An immutable string containing the characters from *data*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

In the CFData object form, the string can be written to disk as a file or be sent out over a network. If the encoding of the characters in the data object is Unicode, the function reads any BOM (byte order marker) and properly resolves endianness.

The `CFStringCreateExternalRepresentation` (page 22) function complements this function by creating an "external representation" CFData object from a string.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringCreateWithBytes

Creates a string from a buffer containing characters in a specified encoding.

```
CFStringRef CFStringCreateWithBytes (
   CFAllocatorRef alloc,
   const UInt8 *bytes,
   CFIndex numBytes,
   CFStringEncoding encoding,
   Boolean isExternalRepresentation
);
```

Parameters

alloc

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

bytes

A buffer containing characters in the encoding specified by *encoding*. The buffer must *not* contain a length byte (as in Pascal buffers) or any terminating `NULL` character (as in C buffers).

numBytes
: The number of bytes in *bytes*.

encoding
: The string encoding of the characters in the buffer.

isExternalRepresentation
: `true` if the characters in the byte buffer are in an "external representation" format—that is, whether the buffer contains a BOM (byte order marker). This is usually the case for bytes that are read in from a text file or received over the network. Otherwise, pass `false`.

Return Value
An immutable string, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function handles character data in an "external representation" format by interpreting any BOM (byte order marker) character and performing any necessary byte swapping.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonMDEF
ImageClient
MFSLives
MoreIsBetter
NSLMiniBrowser

Declared In
`CFString.h`

CFStringCreateWithBytesNoCopy

Creates a string from a buffer, containing characters in a specified encoding, that might serve as the backing store for the new string.

```
CFStringRef CFStringCreateWithBytesNoCopy (
    CFAllocatorRef alloc,
    const UInt8 *bytes,
    CFIndex numBytes,
    CFStringEncoding encoding,
    Boolean isExternalRepresentation,
    CFAllocatorRef contentsDeallocator
);
```

Parameters

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

bytes
: A buffer containing characters in the encoding specified by *encoding*. The buffer must *not* contain a length byte (as in Pascal buffers) or any terminating `NULL` character (as in C buffers).

numBytes
: The number of bytes in *bytes*.

encoding
: The character encoding of *bytes*.

isExternalRepresentation
: `true` if the characters in the byte buffer are in an "external representation" format—that is, whether the buffer contains a BOM (byte order marker). This is usually the case for bytes that are read in from a text file or received over the network. Otherwise, pass `false`.

contentsDeallocator
: The allocator to use to deallocate *bytes* when it is no longer needed. You can pass `NULL` or `kCFAllocatorDefault` to request the default allocator for this purpose. If the buffer does not need to be deallocated, or if you want to assume responsibility for deallocating the buffer (and not have the string deallocate it), pass `kCFAllocatorNull`.

Return Value
A new string whose contents are *bytes*. Ownership follows the Create Rule.

Discussion
This function takes an explicit length, and allows you to specify whether the data is an external format—that is, whether to pay attention to the BOM character (if any) and do byte swapping if necessary

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not deallocated. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

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

See Also
`CFStringCreateWithBytes` (page 24)
`CFStringCreateWithCharactersNoCopy` (page 27)
`CFStringCreateWithCStringNoCopy` (page 29)
`CFStringCreateWithPascalStringNoCopy` (page 33)

Declared In
`CFString.h`

CFStringCreateWithCharacters

Creates a string from a buffer of Unicode characters.

```
CFStringRef CFStringCreateWithCharacters (
   CFAllocatorRef alloc,
   const UniChar *chars,
   CFIndex numChars
);
```

Parameters

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

chars
: The buffer of Unicode characters to copy into the new string.

`numChars`
: The number of characters in the buffer pointed to by *`chars`*. Only this number of characters will be copied to internal storage.

Return Value
An immutable string containing *`chars`*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an immutable string from a client-supplied Unicode buffer. You must supply a count of the characters in the buffer. This function always copies the characters in the provided buffer into internal storage.

To save memory, this function might choose to store the characters internally in a 8-bit backing store. That is, just because a buffer of `UniChar` characters was used to initialize the object does not mean you will get back a non-`NULL` result from `CFStringGetCharactersPtr` (page 42).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Custom_HIView_Tutorial

MoreIsBetter

QISA

QTSetMovieAudioDevice

TypeServicesForUnicode

Declared In
`CFString.h`

CFStringCreateWithCharactersNoCopy

Creates a string from a buffer of Unicode characters that might serve as the backing store for the object.

```
CFStringRef CFStringCreateWithCharactersNoCopy (
   CFAllocatorRef alloc,
   const UniChar *chars,
   CFIndex numChars,
   CFAllocatorRef contentsDeallocator
);
```

Parameters

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

`chars`
: The Unicode buffer that has been allocated and initialized with Unicode characters.

`numChars`
: The number of characters in the buffer pointed to by *`chars`*. Only this number of characters will be copied to internal storage.

contentsDeallocator

The allocator to use to deallocate the external buffer when it is no longer needed. You can pass `NULL` or `kCFAllocatorDefault` to request the default allocator for this purpose. If the buffer does not need to be deallocated, or if you want to assume responsibility for deallocating the buffer (and not have the string deallocate it), pass `kCFAllocatorNull`.

Return Value

An immutable string containing *chars*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

Unless the situation warrants otherwise, the returned object does not copy the external buffer to internal storage but instead uses the buffer as its backing store. However, you should never count on the object using the external buffer since it could copy the buffer to internal storage or might even dump the buffer altogether and use alternative means for storing the characters.

The function includes a *contentsDeallocator* parameter with which to specify an allocator to use for deallocating the external buffer when the string is deallocated. If you want to assume responsibility for deallocating this memory, specify `kCFAllocatorNull` for this parameter.

If at creation time CFString decides it can't use the buffer, and there is a *contentsDeallocator*, it will use this allocator to free the buffer at that time.

Special Considerations

If an error occurs during the creation of the string, then *chars* is not deallocated. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFStringCreateWithCharacters` (page 26)

`CFStringCreateWithBytesNoCopy` (page 25)

`CFStringCreateWithCStringNoCopy` (page 29)

`CFStringCreateWithPascalStringNoCopy` (page 33)

Declared In

`CFString.h`

CFStringCreateWithCString

Creates an immutable string from a C string.

```
CFStringRef CFStringCreateWithCString (
   CFAllocatorRef alloc,
   const char *cStr,
   CFStringEncoding encoding
);
```

Parameters

alloc

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

cStr

The `NULL`-terminated C string to be used to create the CFString object. The string must use an 8-bit encoding.

encoding

The encoding of the characters in the C string. The encoding must specify an 8-bit encoding.

Return Value

An immutable string containing *cStr* (after stripping off the `NULL` terminating character), or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

A C string is a string of 8-bit characters terminated with an 8-bit `NULL`. Unichar and Unichar32 are not considered C strings.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFFTPSample

DockBrowser

MoreIsBetter

NSLMiniBrowser

Quartz EB

Declared In

`CFString.h`

CFStringCreateWithCStringNoCopy

Creates a CFString object from an external C string buffer that might serve as the backing store for the object.

```
CFStringRef CFStringCreateWithCStringNoCopy (
   CFAllocatorRef alloc,
   const char *cStr,
   CFStringEncoding encoding,
   CFAllocatorRef contentsDeallocator
);
```

Parameters

alloc

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

cStr

The `NULL`-terminated C string to be used to create the CFString object. The string must use an 8-bit encoding.

encoding

The encoding of the characters in the C string. The encoding must specify an 8-bit encoding.

contentsDeallocator

The CFAllocator object to use to deallocate the external string buffer when it is no longer needed. You can pass `NULL` or `kCFAllocatorDefault` to request the default allocator for this purpose. If the buffer does not need to be deallocated, or if you want to assume responsibility for deallocating the buffer (and not have the CFString object deallocate it), pass `kCFAllocatorNull`.

Return Value
An immutable string containing *cStr* (after stripping off the `NULL` terminating character), or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
A C string is a string of 8-bit characters terminated with an 8-bit `NULL`. Unichar and Unichar32 are not considered C strings.

Unless the situation warrants otherwise, the created object does not copy the external buffer to internal storage but instead uses the buffer as its backing store. However, you should never assume that the object is using the external buffer since the object might copy the buffer to internal storage or even dump the buffer altogether and store the characters in another way.

The function includes a *contentsDeallocator* parameter with which to specify an allocator to use for deallocating the external buffer when the CFString object is deallocated. If you want to assume responsibility for deallocating this memory, specify `kCFAllocatorNull` for this parameter.

If at creation time the CFString object decides it can't use the buffer, and the function specifies a *contentsDeallocator* allocator, it will use this allocator to free the buffer at that time.

Special Considerations

If an error occurs during the creation of the string, then *cStr* is not deallocated. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFStringCreateWithCString` (page 28)
`CFStringCreateWithBytesNoCopy` (page 25)
`CFStringCreateWithCharactersNoCopy` (page 27)
`CFStringCreateWithPascalStringNoCopy` (page 33)

Declared In
`CFString.h`

CFStringCreateWithFileSystemRepresentation

Creates a CFString from a zero-terminated POSIX file system representation.

```
CFStringRef CFStringCreateWithFileSystemRepresentation (
   CFAllocatorRef alloc,
   const char *buffer
);
```

Parameters

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

buffer
: The C string that you want to convert.

Return Value
A string that represents *buffer*. The result is NULL if there was a problem in creating the string (possible if the conversion fails due to bytes in the buffer not being a valid sequence of bytes for the appropriate character encoding). Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CFString.h

CFStringCreateWithFormat

Creates an immutable string from a formatted string and a variable number of arguments.

```
CFStringRef CFStringCreateWithFormat (
   CFAllocatorRef alloc,
   CFDictionaryRef formatOptions,
   CFStringRef format,
   ...
);
```

Parameters

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

formatOptions
A CFDictionary object containing formatting options for the string (such as the thousand-separator character, which is dependent on locale). Currently, these options are an unimplemented feature.

format
The formatted string with printf-style specifiers. For information on supported specifiers, see String Format Specifiers.

...
Variable list of the values to be inserted in *format*.

Return Value
An immutable string, or NULL if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
HID Calibrator

HID Explorer

HID Utilities Source

MoreIsBetter

QISA

Declared In
CFString.h

CFStringCreateWithFormatAndArguments

Creates an immutable string from a formatted string and a variable number of arguments (specified in a parameter of type `va_list`).

```
CFStringRef CFStringCreateWithFormatAndArguments (
   CFAllocatorRef alloc,
   CFDictionaryRef formatOptions,
   CFStringRef format,
   va_list arguments
);
```

Parameters

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

`formatOptions`
: A CFDictionary object containing formatting options for the string (such as the thousand-separator character, which is dependent on locale). Currently, these options are an unimplemented feature.

`format`
: The formatted string with `printf`-style specifiers. For information on supported specifiers, see String Format Specifiers.

`arguments`
: The variable argument list of values to be inserted into the formatted string contained in *`format`*.

Return Value
An immutable string, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
The programming interface for variable argument lists (`va_list`, `va_start`, `va_end`, and so forth) is declared in the standard C header file `stdarg.h`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringCreateWithPascalString

Creates an immutable CFString object from a Pascal string.

```
CFStringRef CFStringCreateWithPascalString (
   CFAllocatorRef alloc,
   ConstStr255Param pStr,
   CFStringEncoding encoding
);
```

Parameters

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

pStr

The Pascal string to be used to create the string.

encoding

The encoding of the characters in the Pascal string.

Return Value

An immutable string containing *pStr*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

This function creates an immutable CFString objects from the character contents of a Pascal string (after stripping off the initial length byte).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AppearanceSampleUpdated

BasicInputMethod

Carbon Porting Tutorial

JustDraw

MoreIsBetter

Declared In

`CFString.h`

CFStringCreateWithPascalStringNoCopy

Creates a CFString object from an external Pascal string buffer that might serve as the backing store for the object.

```
CFStringRef CFStringCreateWithPascalStringNoCopy (
   CFAllocatorRef alloc,
   ConstStr255Param pStr,
   CFStringEncoding encoding,
   CFAllocatorRef contentsDeallocator
);
```

Parameters

alloc

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

pStr

The Pascal string to be used to create the string.

encoding

The encoding of the characters in the Pascal string.

contentsDeallocator

The CFAllocator object to use to deallocate the external string buffer when it is no longer needed. Pass `NULL` or `kCFAllocatorDefault` to request the default allocator for this purpose. If the buffer does not need to be deallocated, or if you want to assume responsibility for deallocating the buffer (and not have the string deallocate it), pass `kCFAllocatorNull`.

Return Value
An immutable string containing *`pStr`*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an immutable CFString objects from the character contents of a Pascal string (after stripping off the initial length byte).

Unless the situation warrants otherwise, the created object does not copy the external buffer to internal storage but instead uses the buffer as its backing store. However, you should never assume that the object is using the external buffer since the object might copy the buffer to internal storage or even dump the buffer altogether and store the characters in another way.

The function includes a *`contentsDeallocator`* parameter with which to specify an allocator to use for deallocating the external buffer when the string is deallocated. If you want to assume responsibility for deallocating this memory, specify `kCFAllocatorNull` for this parameter.

If at creation time the string decides it can't use the buffer, and there is an allocator specified in the *`contentsDeallocator`* parameter, it will use this allocator to free the buffer at that time.

Special Considerations

If an error occurs during the creation of the string, then *`pStr`* is not deallocated. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFStringCreateWithPascalString` (page 32)
`CFStringCreateWithBytesNoCopy` (page 25)
`CFStringCreateWithCStringNoCopy` (page 29)
`CFStringCreateWithCharactersNoCopy` (page 27)

Declared In
`CFString.h`

CFStringCreateWithSubstring

Creates an immutable string from a segment (substring) of an existing string.

```
CFStringRef CFStringCreateWithSubstring (
   CFAllocatorRef alloc,
   CFStringRef str,
   CFRange range
);
```

Parameters

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

`str`
    The string from which to create the new string.

range

The range of characters in `str` to copy. The specified range must not exceed the length of the string.

Return Value

An immutable string, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioQueueTools

DockBrowser

MoreIsBetter

MoreSCF

TypeServicesForUnicode

Declared In

`CFString.h`

CFStringFind

Searches for a substring within a string and, if it is found, yields the range of the substring within the object's characters.

```
CFRange CFStringFind (
   CFStringRef theString,
   CFStringRef stringToFind,
   CFOptionFlags compareOptions
);
```

Parameters

theString

The string in which to search for *stringToFind*.

stringToFind

The string to search for in *theString*.

compareOptions

Flags that select different types of comparisons, such as localized comparison, case-insensitive comparison, and non-literal comparison. If you want the default comparison behavior, pass `0`. See "String Comparison Flags" (page 60) for the available flags.

Return Value

The range of the located substring within *theString*. If a match is not located, the returned `CFRange` structure will have a location of `kCFNotFound` and a length of *0* (either of which is enough to indicate failure).

Discussion

This function is a convenience when you want to know if the entire range of characters represented by a string contains a particular substring. If you want to search only part of the characters of a string, use the `CFStringFindWithOptions` (page 37) function. Both of these functions return upon finding the first occurrence of the substring, so if you want to find out about multiple occurrences, call the `CFStringCreateArrayWithFindResults` (page 20) function.

Depending on the comparison-option flags specified, the length of the resulting range might be different than the length of the search string.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioQueueTools

CIVideoDemoGL

TypeServicesForUnicode

Declared In
`CFString.h`

CFStringFindCharacterFromSet

Query the range of the first character contained in the specified character set.

```
Boolean CFStringFindCharacterFromSet (
   CFStringRef theString,
   CFCharacterSetRef theSet,
   CFRange rangeToSearch,
   CFOptionFlags searchOptions,
   CFRange *result
);
```

Parameters

theString
: The string to search.

theSet
: The character set against which the membership of characters is checked.

rangeToSearch
: The range of characters within *theString* to search. If the range location or end point (defined by the location plus length minus `1`) are outside the index space of the string (`0` to `N-1` inclusive, where `N` is the length of the string), the behavior is undefined. The specified range must not exceed the length of the string. If the range length is negative, the behavior is undefined. The range may be empty (length `0`), in which case no search is performed.

searchOptions
: The option flags to control the search behavior. The supported options are `kCFCompareBackwards` (page 60) and `kCFCompareAnchored` (page 60). If other option flags are specified, the behavior is undefined.

result
: On return, a pointer to a CFRange structure (supplied by the caller) in which the search result is stored. Note that the length of this range could be more than `1` (if the character in question is a multi-byte character). If a pointer to an invalid structure is passed, the behavior is undefined.

Return Value
`true` if a character in the character set is found and *result* is filled, `false` otherwise.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CFString.h`

CFStringFindWithOptions

Searches for a substring within a range of the characters represented by a string and, if the substring is found, returns its range within the object's characters.

```
Boolean CFStringFindWithOptions (
   CFStringRef theString,
   CFStringRef stringToFind,
   CFRange rangeToSearch,
   CFOptionFlags searchOptions,
   CFRange *result
);
```

Parameters

theString
: The string in which to to search for *stringToFind*.

stringToFind
: The substring to search for in *theString*.

rangeToSearch
: A range of the characters to search in *theString*. The specified range must not exceed the length of the string.

searchOptions
: The option flags to control the search behavior. The supported options are `kCFCompareBackwards` (page 60), `kCFCompareAnchored` (page 60), `kCFCompareCaseInsensitive` (page 60), `kCFCompareNonliteral` (page 60), and `kCFCompareLocalized` (page 61) (available in Mac OS X v10.0 and later). Uses the current user locale (the return value from `CFLocaleCopyCurrent`) if `kCFCompareLocalized` is specified. If other option flags are specified, the behavior is undefined.

result
: On return, if the function result is `true`, contains the starting location and length of the found substring. You may pass `NULL` if you only want to know if the substring exists in the larger string.

Return Value

`true` if the substring was found, `false` otherwise.

Discussion

This function allows you to search only part of the characters of a string for a substring. It returns the found range indirectly, in the final *result* parameter. If you want to know if the entire range of characters represented by a string contains a particular substring, you can use the convenience function `CFStringFind` (page 35). Both of these functions return upon finding the first occurrence of the substring, so if you want to find out about multiple occurrences, call the `CFStringCreateArrayWithFindResults` (page 20) function.

Depending on the comparison-option flags specified, the length of the resulting range might be different than the length of the search string.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DockBrowser

MoreIsBetter

MoreOSL

MoreSCF
QISA

Declared In
`CFString.h`

CFStringFindWithOptionsAndLocale

Returns a Boolean value that indicates whether a given string was found in a given source string.

```
Boolean CFStringFindWithOptionsAndLocale (
   CFStringRef theString,
   CFStringRef stringToFind,
   CFRange rangeToSearch,
   CFOptionFlags searchOptions,
   CFLocaleRef locale,
   CFRange *result
);
```

Parameters

`theString`
The string in which to to search for *`stringToFind`*.

`stringToFind`
The substring to search for in *`theString`*.

`rangeToSearch`
A range of the characters to search in *`theString`*. The specified range must not exceed the length of the string.

`searchOptions`
The option flags to control the search behavior. See "`String Comparison Flags`" (page 60) for possible values. `kCFCompareNumerically` (page 61) is ignored.

`locale`
The locale to use for the search comparison. `NULL` specifies the canonical locale (the return value from `CFLocaleGetSystem`).

The locale argument affects the equality checking algorithm. For example, for the Turkish locale, case-insensitive compare matches "I" to "ı" (Unicode code point U+0131, Latin Small Dotless I), not the normal "i" character.

`result`
On return, if the function result is `true` contains the starting location and length of the found substring. You may pass `NULL` if you only want to know if the *`theString`* contains *`stringToFind`*.

Return Value
`true` if the substring was found, `false` otherwise.

Discussion
If *`stringToFind`* is the empty string (zero length), nothing is found.

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

Declared In
`CFString.h`

CFStringGetBytes

Fetches a range of the characters from a string into a byte buffer after converting the characters to a specified encoding.

```
CFIndex CFStringGetBytes (
   CFStringRef theString,
   CFRange range,
   CFStringEncoding encoding,
   UInt8 lossByte,
   Boolean isExternalRepresentation,
   UInt8 *buffer,
   CFIndex maxBufLen,
   CFIndex *usedBufLen
);
```

Parameters

theString

The string upon which to operate.

range

The range of characters in *theString* to process. The specified range must not exceed the length of the string.

encoding

The string encoding of the characters to copy to the byte buffer. 8, 16, and 32-bit encodings are supported.

lossByte

A character (for example, '?') that should be substituted for characters that cannot be converted to the specified encoding. Pass `0` if you do not want lossy conversion to occur.

isExternalRepresentation

`true` if you want the result to be in an “external representation” format, otherwise `false`. In an “external representation” format, the result may contain a byte order marker (BOM) specifying endianness and this function might have to perform byte swapping.

buffer

The byte buffer into which the converted characters are written. The buffer can be allocated on the heap or stack. Pass `NULL` if you do not want conversion to take place but instead want to know if conversion will succeed (the function result is greater than `0`) and, if so, how many bytes are required (*usedBufLen*).

maxBufLen

The size of *buffer* and the maximum number of bytes that can be written to it.

usedBufLen

On return, the number of converted bytes actually in *buffer*. You may pass `NULL` if you are not interested in this information.

Return Value

The number of characters converted.

Discussion

This function is the basic encoding-conversion function for CFString objects. As with the other functions that get the character contents of CFString objects, it allows conversion to a supported 8-bit encoding. Unlike most of those other functions, it also allows “lossy conversion.” The function permits the specification of a “loss byte” in a parameter; if a character cannot be converted this character is substituted and conversion proceeds. (With the other functions, conversion stops at the first error and the operation fails.)

Because this function takes a range and returns the number of characters converted, it can be called repeatedly with a small fixed size buffer and different ranges of the string to do the conversion incrementally.

This function also handles any necessary manipulation of character data in an "external representation" format. This format makes the data portable and persistent (disk-writable); in Unicode it often includes a BOM (byte order marker) that specifies the endianness of the data.

The `CFStringCreateExternalRepresentation` (page 22) function also handles external representations and performs lossy conversions. The complementary function `CFStringCreateWithBytes` (page 24) creates a string from the characters in a byte buffer.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
audioburntest

bulkerase

databurntest

MoreIsBetter

QISA

Declared In
`CFString.h`

CFStringGetCharacterAtIndex

Returns the Unicode character at a specified location in a string.

```
UniChar CFStringGetCharacterAtIndex (
   CFStringRef theString,
   CFIndex idx
);
```

Parameters

theString
    The string from which the Unicode character is obtained.

idx
    The position of the Unicode character in the CFString.

Return Value
A Unicode character.

Discussion
This function is typically called in a loop to fetch the Unicode characters of a string in sequence or to fetch a character at a known position (first or last, for example). Using it in a loop can be inefficient, especially with longer strings, so consider the `CFStringGetCharacters` (page 41) function or the in-line buffer functions (`CFStringInitInlineBuffer` (page 57) and `CFStringGetCharacterFromInlineBuffer` (page 41)) as alternatives.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AuthForAll

MFSLives
MoreSCF

Declared In
`CFString.h`

CFStringGetCharacterFromInlineBuffer

Returns the Unicode character at a specific location in an in-line buffer.

```
UniChar CFStringGetCharacterFromInlineBuffer (
   CFStringInlineBuffer *buf,
   CFIndex idx
);
```

Parameters

buf
: The initialized `CFStringInlineBuffer` (page 59) structure in which the characters are stored. You should initialize the structure with the `CFStringInitInlineBuffer` (page 57) function.

idx
: The location of a character in the in-line buffer *buf*. This index is relative to the range specified when *buf* was created.

Return Value
A Unicode character, or `0` if a location outside the original range is specified.

Discussion
This function accesses one of the characters of a string written to an in-line buffer. It is typically called from within a loop to access each character in the buffer in sequence. You should initialize the buffer with the `CFStringInitInlineBuffer` (page 57) function. The in-line buffer functions, along with the `CFStringInlineBuffer` (page 59) structure, give you fast access to the characters of a CFString object. The technique for in-line buffer access combines the convenience of one-at-a-time character access with the efficiency of bulk access.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringGetCharacters

Copies a range of the Unicode characters from a string to a user-provided buffer.

```
void CFStringGetCharacters (
   CFStringRef theString,
   CFRange range,
   UniChar *buffer
);
```

Parameters

theString
: The string from which the characters are to be obtained.

range
: The range of characters to copy. The specified range must not exceed the length of the string.

buffer
: The `UniChar` buffer of length `range.length` that you have allocated on the stack or heap. On return, the buffer contains the requested Unicode characters.

Discussion
Use this function to obtain some or all of the Unicode characters represented by a CFString object. If this operation involves a large number of characters, the function call can be expensive in terms of memory. Instead you might want to consider using the in-line buffer functions `CFStringInitInlineBuffer` (page 57) and `CFStringGetCharacterFromInlineBuffer` (page 41) to extract the characters incrementally.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQuartzDrawingWPrinting
Custom_HIView_Tutorial
HITextViewDemo
MoreIsBetter
QISA

Declared In
`CFString.h`

CFStringGetCharactersPtr

Quickly obtains a pointer to the contents of a string as a buffer of Unicode characters.

```
const UniChar * CFStringGetCharactersPtr (
   CFStringRef theString
);
```

Parameters

theString
: The string whose contents you wish to access.

Return Value
A pointer to a buffer of Unicode character, or `NULL` if the internal storage of *theString* does not allow this to be returned efficiently.

Discussion
This function either returns the requested pointer immediately, with no memory allocations and no copying, or it returns `NULL`. If the latter is the result, call an alternative function such as `CFStringGetCharacters` (page 41) function to extract the characters.

Whether or not this function returns a valid pointer or `NULL` depends on many factors, all of which depend on how the string was created and its properties. In addition, the function result might change between different releases and on different platforms. So do not count on receiving a non-`NULL` result from this function under any circumstances (except when the object is created with `CFStringCreateMutableWithExternalCharactersNoCopy`).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
TypeServicesForUnicode

Declared In
`CFString.h`

CFStringGetCString

Copies the character contents of a string to a local C string buffer after converting the characters to a given encoding.

```
Boolean CFStringGetCString (
   CFStringRef theString,
   char *buffer,
   CFIndex bufferSize,
   CFStringEncoding encoding
);
```

Parameters

`theString`
The string whose contents you wish to access.

`buffer`
The C string buffer into which to copy the string. The buffer must be at least *`bufferSize`* bytes in length. On return, the buffer contains the converted characters. If there is an error in conversion, the buffer contains only partial results.

`bufferSize`
The length of the local *`buffer`* in bytes (accounting for the `NULL`-terminator byte).

`encoding`
The string encoding to which the character contents of *`theString`* should be converted. The encoding must specify an 8-bit encoding.

Return Value
`true` upon success or `false` if the conversion fails or the provided buffer is too small.

Discussion
This function is useful when you need your own copy of a string's character data as a C string. You also typically call it as a "backup" when a prior call to the `CFStringGetCStringPtr` (page 44) function fails.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CFHostSample
HID Calibrator
HID Config Save
HID Explorer
MoreIsBetter

Declared In
`CFString.h`

CFStringGetCStringPtr

Quickly obtains a pointer to a C-string buffer containing the characters of a string in a given encoding.

```
const char * CFStringGetCStringPtr (
   CFStringRef theString,
   CFStringEncoding encoding
);
```

Parameters

theString

The string whose contents you wish to access.

encoding

The string encoding to which the character contents of *theString* should be converted. The encoding must specify an 8-bit encoding.

Return Value

A pointer to a C string or `NULL` if the internal storage of *theString* does not allow this to be returned efficiently.

Discussion

This function either returns the requested pointer immediately, with no memory allocations and no copying, in constant time, or returns `NULL`. If the latter is the result, call an alternative function such as the `CFStringGetCString` (page 43) function to extract the characters.

Whether or not this function returns a valid pointer or `NULL` depends on many factors, all of which depend on how the string was created and its properties. In addition, the function result might change between different releases and on different platforms. So do not count on receiving a non-`NULL` result from this function under any circumstances.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ColorMatching

ColorSyncDevices

ColorSyncDevices-Cocoa

InkSample

NSLMiniBrowser

Declared In

`CFString.h`

CFStringGetDoubleValue

Returns the primary `double` value represented by a string.

```
double CFStringGetDoubleValue (
   CFStringRef str
);
```

Parameters

str

A string that represents a double value. The only allowed characters are the ASCII digit characters (ASCII `0x30` - `0x39`), the plus sign (ASCII `0x2B`), the minus sign (ASCII `0x2D`), and the period character (ASCII `0x2E`).

Return Value

The `double` value represented by *str*, or `0.0` if there is a scanning error (if the string contains disallowed characters or does not represent a double value).

Discussion

Consider the following example:

```
double val = CFStringGetDoubleValue(CFSTR("0.123"));
```

The variable `val` in this example would contain the value `0.123` after the function is called.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringGetFastestEncoding

Returns for a CFString object the character encoding that requires the least conversion time.

```
CFStringEncoding CFStringGetFastestEncoding (
   CFStringRef theString
);
```

Parameters

theString

The string for which to determine the fastest encoding.

Return Value

The string encoding to which *theString* can be converted the fastest.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringGetFileSystemRepresentation

Extracts the contents of a string as a `NULL`-terminated 8-bit string appropriate for passing to POSIX APIs.

```
Boolean CFStringGetFileSystemRepresentation (
   CFStringRef string,
   char *buffer,
   CFIndex maxBufLen
);
```

Parameters

string

The string to convert.

buffer

The C string buffer into which to copy the string. The buffer must be at least *maxBufLen* bytes in length. On return, the buffer contains the converted characters.

maxBufLen

The maximum length of the buffer.

Return Value

true if the string is correctly converted; false if the conversion fails, or the results don't fit into the buffer.

Discussion

You can use CFStringGetMaximumSizeOfFileSystemRepresentation (page 49) if you want to make sure the buffer is of sufficient length.

Availability

Available in Mac OS X v10.4 and later.

Declared In

CFString.h

CFStringGetIntValue

Returns the integer value represented by a string.

```
SInt32 CFStringGetIntValue (
   CFStringRef str
);
```

Parameters

str

A string that represents a signed integer value. The only allowed characters are the ASCII digit characters (ASCII 0x30 - 0x39), the plus sign (ASCII 0x2B), the minus sign (ASCII 0x2D), and the period character (ASCII 0x2E).

Return Value

The signed integer value represented by *str*. The result is 0 if there is a scanning error (if the string contains disallowed characters or does not represent an integer value) or INT_MAX or INT_MIN if there is an overflow error.

Discussion

Consider the following example:

```
SInt32 val = CFStringGetIntValue(CFSTR("-123"));
```

The variable val in this example would contain the value -123 after the function is called.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringGetLength

Returns the number (in terms of UTF-16 code pairs) of Unicode characters in a string.

```
CFIndex CFStringGetLength (
   CFStringRef theString
);
```

Parameters

`theString`
    The string to examine.

Return Value
The number (in terms of UTF-16 code pairs) of characters stored in *`theString`*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Custom_HIView_Tutorial

MoreIsBetter

MoreSCF

QISA

TypeServicesForUnicode

Declared In
`CFString.h`

CFStringGetLineBounds

Given a range of characters in a string, obtains the line bounds—that is, the indexes of the first character and the final characters of the lines containing the range.

```
void CFStringGetLineBounds (
   CFStringRef theString,
   CFRange range,
   CFIndex *lineBeginIndex,
   CFIndex *lineEndIndex,
   CFIndex *contentsEndIndex
);
```

Parameters

`theString`
    The string containing the specified range of characters.

`range`
    The range of characters to consider. The specified range must not exceed the length of the string.

`lineBeginIndex`
: On return, the index of the first character of the containing line. Pass `NULL` if you do not want this result.

`lineEndIndex`
: On return, the index of the first character of the line after the specified range. Pass `NULL` if you do not want this result.

`contentsEndIndex`
: On return, the index of the last character of the containing line, excluding any line-separator characters. Pass `NULL` if you are not interested in this result.

Discussion

This function is a convenience function for determining the beginning and ending indexes of one or more lines in the given range of a string. It is useful, for example, when each line represents a “record” of some sort; you might search for some substring, but want to extract the record of which the substring is a part.

To determine line separation, the function looks for the standard line-separator characters: carriage returns (CR and CRLF), linefeeds (LF), and Unicode line and paragraph separators. The three final parameters of the function indirectly return, in order, the index of the first character that starts the line, the index of the first character of the next line (including end-of-line characters), and the index of the last character of the line (excluding end-of-line characters). Pass `NULL` for any of these parameters if you aren't interested in the result.

To determine the number of characters in the line:

- Subtract *`lineBeginIndex`* from *`lineEndIndex`* to find the number of characters in the line, including the line separators.
- Subtract *`lineBeginIndex`* from *`contentsEndIndex`* to find the number of characters in the line, excluding the line separators.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringGetListOfAvailableEncodings

Returns a pointer to a list of string encodings supported by the current system.

```
const CFStringEncoding * CFStringGetListOfAvailableEncodings (
   void
);
```

Return Value

A pointer to a `kCFStringEncodingInvalidId` (page 64)-terminated list of `enum` constants, each of type `CFStringEncoding` (page 58).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
`CFString.h`

CFStringGetMaximumSizeForEncoding

Returns the maximum number of bytes a string of a specified length (in Unicode characters) will take up if encoded in a specified encoding.

```
CFIndex CFStringGetMaximumSizeForEncoding (
   CFIndex length,
   CFStringEncoding encoding
);
```

Parameters

length
: The number of Unicode characters to evaluate.

encoding
: The string encoding for the number of characters specified by *length*.

Return Value
The maximum number of bytes that could be required to represent *length* number of Unicode characters with the string encoding *encoding*. The number of bytes that the encoding actually ends up requiring when converting any particular string could be less than this, but never more.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CFHostSample
IOPrintSuperClasses
MoreIsBetter
simpleJavaLauncher

Declared In
`CFString.h`

CFStringGetMaximumSizeOfFileSystemRepresentation

Determines the upper bound on the number of bytes required to hold the file system representation of the string.

```
CFIndex CFStringGetMaximumSizeOfFileSystemRepresentation (
   CFStringRef string
);
```

Parameters

string
: The string to convert.

Return Value
The upper bound on the number of bytes required to hold the file system representation of the string.

Discussion
The result is returned quickly as a rough approximation, and could be much larger than the actual space required. The result includes space for the zero termination. If you are allocating a buffer for long-term storage, you should reallocate it to be the right size after calling `CFStringGetFileSystemRepresentation` (page 45).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CFString.h`

CFStringGetMostCompatibleMacStringEncoding

Returns the most compatible Mac OS script value for the given input encoding.

```
CFStringEncoding CFStringGetMostCompatibleMacStringEncoding (
   CFStringEncoding encoding
);
```

Parameters

`encoding`
The encoding for which you wish to find a compatible Mac OS script value.

Return Value
The most compatible Mac OS script value for *`encoding`*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`CFString.h`

CFStringGetNameOfEncoding

Returns the canonical name of a specified string encoding.

```
CFStringRef CFStringGetNameOfEncoding (
   CFStringEncoding encoding
);
```

Parameters

`encoding`
The string encoding to use.

Return Value
Name of *`encoding`*; non-localized. Ownership follows the Get Rule.

Discussion
This function returns the "canonical" name of the string encoding because the return value has to be the same no matter what localization is chosen. In other words, it can't change based on the International Preferences language panel setting. The canonical name is usually expressed in English.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringGetParagraphBounds

Given a range of characters in a string, obtains the paragraph bounds—that is, the indexes of the first character and the final characters of the paragraph(s) containing the range.

```
void CFStringGetParagraphBounds (
   CFStringRef string,
   CFRange range,
   CFIndex *parBeginIndex,
   CFIndex *parEndIndex,
   CFIndex *contentsEndIndex
);
```

Parameters

theString
The string containing the specified range of characters.

range
The range of characters to consider. The specified range must not exceed the length of the string.

parBeginIndex
On return, the index of the first character of the containing paragraph. Pass `NULL` if you do not want this result.

parEndIndex
On return, the index of the first character of the paragraph after the specified range. Pass `NULL` if you do not want this result.

contentsEndIndex
On return, the index of the last character of the containing paragraph, excluding any paragraph-separator characters. Pass `NULL` if you are not interested in this result.

Discussion
This function is the same as `CFStringGetLineBounds` (page 47)(), however it onlys look for paragraphs (that is, it does not stop at Unicode NextLine or LineSeparator characters).

This function is a convenience function for determining the beginning and ending indexes of one or more paragraph in the given range of a string. It is useful, for example, when each line represents a "record" of some sort; you might search for some substring, but want to extract the record of which the substring is a part.

To determine line separation, the function looks for the standard paragraph-separator characters: carriage returns (CR and CRLF), linefeeds (LF), and Unicode paragraph separators. The three final parameters of the function indirectly return, in order, the index of the first character that starts the line, the index of the first character of the next line (including end-of-line characters), and the index of the last character of the line (excluding end-of-line characters). Pass `NULL` for any of these parameters if you aren't interested in the result.

To determine the number of characters in the paragraph:

- Subtract *parBeginIndex* from *parEndIndex* to find the number of characters in the paragraph, including the paragraph separators.
- Subtract *parBeginIndex* from *contentsEndIndex* to find the number of characters in the paragraph, excluding the paragraph separators.

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

Declared In
`CFString.h`

CFStringGetPascalString

Copies the character contents of a CFString object to a local Pascal string buffer after converting the characters to a requested encoding.

```
Boolean CFStringGetPascalString (
   CFStringRef theString,
   StringPtr buffer,
   CFIndex bufferSize,
   CFStringEncoding encoding
);
```

Parameters

theString

The string to examine.

buffer

The Pascal string buffer into which to copy the *theString*. The buffer must be at least *bufferSize* bytes in length. On return, contains the converted characters. If there is an error in conversion, the buffer contains only partial results.

bufferSize

The length of the local *buffer* in bytes (accounting for the length byte).

encoding

The string encoding to which the character contents of *theString* should be converted.

Return Value
`true` if the operation succeeds or `false` if the conversion fails or the provided buffer is too small.

Discussion
This function is useful when you need your own copy of a CFString object's character data as a Pascal string. You can also call it as a “backup” operation when a prior call to the `CFStringGetPascalStringPtr` (page 53) function fails.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GrabBag

MoreIsBetter

QISA

QTKitTimeCode

Declared In
`CFString.h`

CFStringGetPascalStringPtr

Quickly obtains a pointer to a Pascal buffer containing the characters of a string in a given encoding.

```
ConstStringPtr CFStringGetPascalStringPtr (
   CFStringRef theString,
   CFStringEncoding encoding
);
```

Parameters

`theString`
: The string to examine.

`encoding`
: The string encoding to which the character contents of *`theString`* should be converted.

Return Value
A pointer to a Pascal string buffer or `NULL` if the internal storage of `theString` does not allow this to be returned efficiently.

Discussion
This function either returns the requested pointer immediately, with no memory allocations and no copying, in constant time, or returns `NULL`. If the latter is returned, call an alternative function such as the `CFStringGetPascalString` (page 52) function to extract the characters.

Whether or not this function returns a valid pointer or `NULL` depends on many factors, all of which depend on how the string was created and its properties. In addition, the function result might change between different releases and on different platforms. So do not count on receiving a non-`NULL` result from this function under any circumstances.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GrabBag

Declared In
`CFString.h`

CFStringGetRangeOfComposedCharactersAtIndex

Returns the range of the composed character sequence at a specified index.

```
CFRange CFStringGetRangeOfComposedCharactersAtIndex (
   CFStringRef theString,
   CFIndex theIndex
);
```

Parameters

theString
    The string to examine.

theIndex
    The index of the character contained in the composed character sequence. If the index is outside the range of the string (`0` to `N-1` inclusive, where `N` is the length of the string), the behavior is undefined.

Return Value
The range of the composed character sequence.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CFString.h`

CFStringGetSmallestEncoding

Returns the smallest encoding on the current system for the character contents of a string.

```
CFStringEncoding CFStringGetSmallestEncoding (
   CFStringRef theString
);
```

Parameters

theString
    The string for which to find the smallest encoding.

Return Value
The string encoding that has the smallest representation of *theString*.

Discussion
This function returns the supported encoding that requires the least space (in terms of bytes needed to represent one character) to represent the character contents of a string. This information is not always immediately available, so this function might need to compute it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringGetSystemEncoding

Returns the default encoding used by the operating system when it creates strings.

```
CFStringEncoding CFStringGetSystemEncoding (
   void
);
```

Return Value
The default string encoding.

Discussion
This function returns the default text encoding used by the OS when it creates strings. In Mac OS X, this encoding is determined by the user's preferred language setting. The preferred language is the first language listed in the International pane of the System Preferences.

In most situations you will not want to use this function, however, because your primary interest will be your application's default text encoding. The application encoding is required when you create a CFStringRef from strings stored in Resource Manager resources, which typically use one of the Mac encodings such as MacRoman or MacJapanese.

To get your application's default text encoding, call the `GetApplicationTextEncoding` Carbon function.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

HID Utilities Source

NSLMiniBrowser

QISA

Declared In
`CFString.h`

CFStringGetTypeID

Returns the type identifier for the CFString opaque type.

```
CFTypeID CFStringGetTypeID (
   void
);
```

Return Value
The type identifier for the CFString opaque type.

Discussion
CFMutableString objects have the same type identifier as CFString objects.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DRDataBurnCarbonUI

DREraseCarbonUI

MoreIsBetter

MoreSCF

QISA

Declared In
`CFString.h`

CFStringHasPrefix

Determines if the character data of a string begin with a specified sequence of characters.

```
Boolean CFStringHasPrefix (
   CFStringRef theString,
   CFStringRef prefix
);
```

Parameters

theString
: The string to search.

prefix
: The prefix to search for.

Return Value
`true` if *theString* begins with *prefix*, `false` if otherwise.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AppleScriptRunner

HID Calibrator

MoreIsBetter

MoreSCF

QISA

Declared In
`CFString.h`

CFStringHasSuffix

Determines if a string ends with a specified sequence of characters.

```
Boolean CFStringHasSuffix (
   CFStringRef theString,
   CFStringRef suffix
);
```

Parameters

theString
: The string to be evaluated.

suffix
: The suffix to search for.

Return Value
`true` if *theString* ends with *suffix*, `false` otherwise.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
HID Calibrator

HID Config Save

HID Explorer

MoreIsBetter

QISA

Declared In
`CFString.h`

CFStringInitInlineBuffer

Initializes an in-line buffer to use for efficient access of a CFString object's characters.

```
void CFStringInitInlineBuffer (
   CFStringRef str,
   CFStringInlineBuffer *buf,
   CFRange range
);
```

Parameters

`str`

The string to copy to the in-line buffer.

`buf`

The (uninitialized) `CFStringInlineBuffer` (page 59) structure to initialize. On return, an initialized structure that can be used in a `CFStringGetCharacterFromInlineBuffer` (page 41) function call. Typically this buffer is allocated on the stack.

`range`

The range of characters in *`str`* to copy to *`buf`*. The specified range must not exceed the length of the string.

Discussion
This function initializes an `CFStringInlineBuffer` (page 59) structure that can be used for accessing the characters of a string. Once the buffer is initialized you can call the `CFStringGetCharacterFromInlineBuffer` (page 41) function to access the characters in the buffer one at a time. The in-line buffer functions, along with the `CFStringInlineBuffer` (page 59) structure, give you fast access to the characters of a string. The technique for in-line buffer access combines the convenience of one-at-a-time character access with the efficiency of bulk access.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFString.h`

CFStringIsEncodingAvailable

Determines whether a given Core Foundation string encoding is available on the current system.

```
Boolean CFStringIsEncodingAvailable (
   CFStringEncoding encoding
);
```

Parameters

encoding

The Core Foundation string encoding to test.

Return Value

`true` if the encoding is available, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

Data Types

CFStringCompareFlags

A `CFOptionFlags` type for specifying options for string comparison .

```
typedef CFOptionFlags CFStringCompareFlags;
```

Discussion

See "String Comparison Flags" (page 60) for values.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringEncoding

An integer type for constants used to specify supported string encodings in various CFString functions.

```
typedef UInt32 CFStringEncoding;
```

Discussion

This type is used to define the constants for the built-in encodings (see "Built-in String Encodings" (page 61) for a list) and for platform-dependent encodings (see "External String Encodings" (page 64)). If CFString does not recognize or support the string encoding of a particular string, CFString functions will identify the string's encoding as `kCFStringEncodingInvalidId` (page 64).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFString.h`

CFStringEncodings

Index type for constants used to specify external string encodings.

```
typedef CFIndex CFStringEncodings;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFStringEncodingExt.h`

CFStringInlineBuffer

Defines the buffer and related fields used for in-line buffer access of characters in CFString objects.

```
struct CFStringInlineBuffer {
    UniChar buffer[64];
    CFStringRef theString;
    const UniChar *directBuffer;
    CFRange rangeToBuffer;
    CFIndex bufferedRangeStart;
    CFIndex bufferedRangeEnd;
};
```

Discussion
This structure is used for in-line buffer access of characters contained by a CFString object. Use the `CFStringInitInlineBuffer` (page 57) function for initializing the fields of this structure; do not do it manually. Once the buffer is initialized, use the `CFStringGetCharacterFromInlineBuffer` (page 41) function to access characters from the buffer. Do not access the fields directly as they might change between releases.

The only reason this structure is not opaque is to allow the in-line functions to access its fields.

Declared In
`CFString.h`

CFStringRef

A reference to a CFString object.

```
typedef const struct __CFString *CFStringRef;
```

Discussion
The `CFStringRef` type refers to a CFString object, which “encapsulates” a Unicode string along with its length. CFString is an opaque type that defines the characteristics and behavior of CFString objects.

Values of type `CFStringRef` may refer to immutable or mutable strings, as CFMutableString objects respond to all functions intended for immutable CFString objects. Functions which accept `CFStringRef` values, and which need to hold on to the values immutably, should call `CFStringCreateCopy` (page 21) (instead of `CFRetain`) to do so.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFBase.h`

Constants

String Comparison Flags

Flags that specify how string comparisons are performed.

```
enum CFStringCompareFlags {
    kCFCompareCaseInsensitive = 1,
    kCFCompareBackwards = 4,
    kCFCompareAnchored = 8,
    kCFCompareNonliteral = 16,
    kCFCompareLocalized = 32,
    kCFCompareNumerically = 64,
    kCFCompareDiacriticInsensitive = 128,
    kCFCompareWidthInsensitive = 256,
    kCFCompareForcedOrdering = 512
};
```

Constants

`kCFCompareCaseInsensitive`

Specifies that the comparison should ignore differences in case between alphabetical characters.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFCompareBackwards`

Specifies that the comparison should start at the last elements of the entities being compared (for example, strings or arrays).

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFCompareAnchored`

Performs searching only on characters at the beginning or end of the range.

No match at the beginning or end means nothing is found, even if a matching sequence of characters occurs elsewhere in the string.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFCompareNonliteral`

Specifies that loose equivalence is acceptable, especially as pertains to diacritical marks.

For example, "ö" represented as two distinct characters ("o" and "umlaut") is equivalent to "ö" represented by a single character ("o-umlaut"). Note that this is not the same as diacritic insensitivity.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFCompareLocalized`
: Specifies that the comparison should take into account differences related to locale, such as the thousands separator character.

  Available in Mac OS X v10.0 and later.

  Declared in `CFString.h`.

`kCFCompareNumerically`
: Specifies that represented numeric values should be used as the basis for comparison and not the actual character values.

  For example, “version 2” is less than “version 2.5”.

  This comparison does not work if `kCFCompareLocalized` is specified on systems before Mac OS X v10.3.

  Available in Mac OS X v10.0 and later.

  Declared in `CFString.h`.

`kCFCompareDiacriticInsensitive`
: Specifies that the comparison should ignore diacritic markers.

  For example, “ö” (“o-umlaut”) is equivalent to “o”.

  Diacritic markers are designated as all non-spacing marks below `U+0510`.

  Available in Mac OS X v10.5 and later.

  Declared in `CFString.h`.

`kCFCompareWidthInsensitive`
: Specifies that the comparison should ignore width differences.

  For example, “a” is equivalent to `UFF41`.

  Available in Mac OS X v10.5 and later.

  Declared in `CFString.h`.

`kCFCompareForcedOrdering`
: Specifies that the comparison is forced to return either `kCFCompareLessThan` or `kCFCompareGreaterThan` if the strings are equivalent but not strictly equal.

  You use this option for stability when sorting (for example, with `kCFCompareCaseInsensitive` specified “aaa” is greater than “AAA”).

  Available in Mac OS X v10.5 and later.

  Declared in `CFString.h`.

Discussion
These constants are flags intended for use in the comparison-option parameters in comparison functions such as `CFStringCompare` (page 13). If you want to request multiple options, combine them with a bitwise-OR operation.

Declared In
`CFString.h`

Built-in String Encodings

Encodings that are built-in on all platforms on which Mac OS X runs.

```
enum CFStringBuiltInEncodings {
    kCFStringEncodingMacRoman = 0,
    kCFStringEncodingWindowsLatin1 = 0x0500,
    kCFStringEncodingISOLatin1 = 0x0201,
    kCFStringEncodingNextStepLatin = 0x0B01,
    kCFStringEncodingASCII = 0x0600,
    kCFStringEncodingUnicode = 0x0100,
    kCFStringEncodingUTF8 = 0x08000100,
    kCFStringEncodingNonLossyASCII = 0x0BFF,

    kCFStringEncodingUTF16 = 0x0100,
    kCFStringEncodingUTF16BE = 0x10000100,
    kCFStringEncodingUTF16LE = 0x14000100,
    kCFStringEncodingUTF32 = 0x0c000100,
    kCFStringEncodingUTF32BE = 0x18000100,
    kCFStringEncodingUTF32LE = 0x1c000100
};
typedef enum CFStringBuiltInEncodings CFStringBuiltInEncodings;
```

Constants

`kCFStringEncodingMacRoman`

An encoding constant that identifies the Mac Roman encoding.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingWindowsLatin1`

An encoding constant that identifies the Windows Latin 1 encoding (ANSI codepage 1252).

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingISOLatin1`

An encoding constant that identifies the ISO Latin 1 encoding (ISO 8859-1)

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingNextStepLatin`

An encoding constant that identifies the NextStep/OpenStep encoding.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingASCII`

An encoding constant that identifies the ASCII encoding (decimal values 0 through 127).

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingUnicode`

An encoding constant that identifies the Unicode encoding.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingUTF8`

An encoding constant that identifies the UTF 8 encoding.

Available in Mac OS X v10.0 and later.

Declared in `CFString.h`.

`kCFStringEncodingNonLossyASCII`
: An encoding constant that identifies non-lossy ASCII encoding.

  Available in Mac OS X v10.0 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF16`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF16Format encoding (alias of kCFStringEncodingUnicode).

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF16BE`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF16BEFormat encoding. This constant specifies big-endian byte order.

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF16LE`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF16LEFormat encoding. This constant specifies little-endian byte order.

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF32`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF32Format encoding.

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF32BE`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF32BEFormat encoding. This constant specifies big-endian byte order.

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

`kCFStringEncodingUTF32LE`
: An encoding constant that identifies kTextEncodingUnicodeDefault + kUnicodeUTF32LEFormat encoding. This constant specifies little-endian byte order.

  Available in Mac OS X v10.4 and later.

  Declared in `CFString.h`.

Declared In
`CFString.h`

Invalid String Encoding Flag

Special value returned from functions to indicate a string encoding that is not supported or recognized by CFString.

```
#define kCFStringEncodingInvalidId (0xffffffffU)
```

Constants

`kCFStringEncodingInvalidId`

Used as a function result to identify an encoding that is not supported or recognized by CFString.

Available in Mac OS X v10.2 and later.

Declared in `CFString.h`.

Declared In

`CFString.h`

External String Encodings

`CFStringEncoding` constants for encodings that may be supported by CFString.

```
enum {
    kCFStringEncodingMacRoman = 0L,
    kCFStringEncodingMacJapanese = 1,
    kCFStringEncodingMacChineseTrad = 2,
    kCFStringEncodingMacKorean = 3,
    kCFStringEncodingMacArabic = 4,
    kCFStringEncodingMacHebrew = 5,
    kCFStringEncodingMacGreek = 6,
    kCFStringEncodingMacCyrillic = 7,
    kCFStringEncodingMacDevanagari = 9,
    kCFStringEncodingMacGurmukhi = 10,
    kCFStringEncodingMacGujarati = 11,
    kCFStringEncodingMacOriya = 12,
    kCFStringEncodingMacBengali = 13,
    kCFStringEncodingMacTamil = 14,
    kCFStringEncodingMacTelugu = 15,
    kCFStringEncodingMacKannada = 16,
    kCFStringEncodingMacMalayalam = 17,
    kCFStringEncodingMacSinhalese = 18,
    kCFStringEncodingMacBurmese = 19,
    kCFStringEncodingMacKhmer = 20,
    kCFStringEncodingMacThai = 21,
    kCFStringEncodingMacLaotian = 22,
    kCFStringEncodingMacGeorgian = 23,
    kCFStringEncodingMacArmenian = 24,
    kCFStringEncodingMacChineseSimp = 25,
    kCFStringEncodingMacTibetan = 26,
    kCFStringEncodingMacMongolian = 27,
    kCFStringEncodingMacEthiopic = 28,
    kCFStringEncodingMacCentralEurRoman = 29,
    kCFStringEncodingMacVietnamese = 30,
    kCFStringEncodingMacExtArabic = 31,
    kCFStringEncodingMacSymbol = 33,
    kCFStringEncodingMacDingbats = 34,
    kCFStringEncodingMacTurkish = 35,
    kCFStringEncodingMacCroatian = 36,
    kCFStringEncodingMacIcelandic = 37,
    kCFStringEncodingMacRomanian = 38,
    kCFStringEncodingMacCeltic = 39,
    kCFStringEncodingMacGaelic = 40,
    kCFStringEncodingMacFarsi = 0x8C,
    kCFStringEncodingMacUkrainian = 0x98,
    kCFStringEncodingMacInuit = 0xEC,
    kCFStringEncodingMacVT100 = 0xFC,
    kCFStringEncodingMacHFS = 0xFF,
    kCFStringEncodingISOLatin1 = 0x0201,
    kCFStringEncodingISOLatin2 = 0x0202,
    kCFStringEncodingISOLatin3 = 0x0203,
    kCFStringEncodingISOLatin4 = 0x0204,
    kCFStringEncodingISOLatinCyrillic = 0x0205,
    kCFStringEncodingISOLatinArabic = 0x0206,
    kCFStringEncodingISOLatinGreek = 0x0207,
    kCFStringEncodingISOLatinHebrew = 0x0208,
    kCFStringEncodingISOLatin5 = 0x0209,
    kCFStringEncodingISOLatin6 = 0x020A,
    kCFStringEncodingISOLatinThai = 0x020B,
    kCFStringEncodingISOLatin7 = 0x020D,
    kCFStringEncodingISOLatin8 = 0x020E,
    kCFStringEncodingISOLatin9 = 0x020F,
    kCFStringEncodingISOLatin10 = 0x0210,
    kCFStringEncodingDOSLatinUS = 0x0400,
    kCFStringEncodingDOSGreek = 0x0405,
    kCFStringEncodingDOSBalticRim = 0x0406,
    kCFStringEncodingDOSLatin1 = 0x0410,
    kCFStringEncodingDOSGreek1 = 0x0411,
    kCFStringEncodingDOSLatin2 = 0x0412,
    kCFStringEncodingDOSCyrillic = 0x0413,
    kCFStringEncodingDOSTurkish = 0x0414,
    kCFStringEncodingDOSPortuguese = 0x0415,
    kCFStringEncodingDOSIcelandic = 0x0416,
    kCFStringEncodingDOSHebrew = 0x0417,
    kCFStringEncodingDOSCanadianFrench = 0x0418,
    kCFStringEncodingDOSArabic = 0x0419,
    kCFStringEncodingDOSNordic = 0x041A,
    kCFStringEncodingDOSRussian = 0x041B,
    kCFStringEncodingDOSGreek2 = 0x041C,
    kCFStringEncodingDOSThai = 0x041D,
    kCFStringEncodingDOSJapanese = 0x0420,
    kCFStringEncodingDOSChineseSimplif = 0x0421,
    kCFStringEncodingDOSKorean = 0x0422,
    kCFStringEncodingDOSChineseTrad = 0x0423,
    kCFStringEncodingWindowsLatin1 = 0x0500,
    kCFStringEncodingWindowsLatin2 = 0x0501,
    kCFStringEncodingWindowsCyrillic = 0x0502,
    kCFStringEncodingWindowsGreek = 0x0503,
    kCFStringEncodingWindowsLatin5 = 0x0504,
    kCFStringEncodingWindowsHebrew = 0x0505,
    kCFStringEncodingWindowsArabic = 0x0506,
    kCFStringEncodingWindowsBalticRim = 0x0507,
    kCFStringEncodingWindowsVietnamese = 0x0508,
    kCFStringEncodingWindowsKoreanJohab = 0x0510,
    kCFStringEncodingASCII = 0x0600,
    kCFStringEncodingANSEL = 0x0601,
    kCFStringEncodingJIS_X0201_76 = 0x0620,
    kCFStringEncodingJIS_X0208_83 = 0x0621,
    kCFStringEncodingJIS_X0208_90 = 0x0622,
    kCFStringEncodingJIS_X0212_90 = 0x0623,
    kCFStringEncodingJIS_C6226_78 = 0x0624,
    kCFStringEncodingShiftJIS_X0213_00 = 0x0628,
    kCFStringEncodingShiftJIS_X0213_MenKuTen = 0x0629,
    kCFStringEncodingGB_2312_80 = 0x0630,
    kCFStringEncodingGBK_95 = 0x0631,
    kCFStringEncodingGB_18030_2000 = 0x0632,
    kCFStringEncodingKSC_5601_87 = 0x0640,
    kCFStringEncodingKSC_5601_92_Johab = 0x0641,
    kCFStringEncodingCNS_11643_92_P1 = 0x0651,
    kCFStringEncodingCNS_11643_92_P2 = 0x0652,
    kCFStringEncodingCNS_11643_92_P3 = 0x0653,
    kCFStringEncodingISO_2022_JP = 0x0820,
    kCFStringEncodingISO_2022_JP_2 = 0x0821,
    kCFStringEncodingISO_2022_JP_1 = 0x0822,
    kCFStringEncodingISO_2022_JP_3 = 0x0823,
    kCFStringEncodingISO_2022_CN = 0x0830,
    kCFStringEncodingISO_2022_CN_EXT = 0x0831,
    kCFStringEncodingISO_2022_KR = 0x0840,
    kCFStringEncodingEUC_JP = 0x0920,
    kCFStringEncodingEUC_CN = 0x0930,
    kCFStringEncodingEUC_TW = 0x0931,
    kCFStringEncodingEUC_KR = 0x0940,
    kCFStringEncodingShiftJIS = 0x0A01,
    kCFStringEncodingKOI8_R = 0x0A02,
    kCFStringEncodingBig5 = 0x0A03,
    kCFStringEncodingMacRomanLatin1 = 0x0A04,
    kCFStringEncodingHZ_GB_2312 = 0x0A05,
    kCFStringEncodingBig5_HKSCS_1999 = 0x0A06,
    kCFStringEncodingVISCII = 0x0A07,
    kCFStringEncodingKOI8_U = 0x0A08,
    kCFStringEncodingBig5_E = 0x0A09,
    kCFStringEncodingNextStepLatin = 0x0B01,
    kCFStringEncodingNextStepJapanese = 0x0B02,
    kCFStringEncodingEBCDIC_US = 0x0C01,
    kCFStringEncodingEBCDIC_CP037 = 0x0C02,
};
```

Constants

`kCFStringEncodingMacJapanese`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacChineseTrad`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacKorean`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacArabic`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacHebrew`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacGreek`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacCyrillic`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacDevanagari`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacGurmukhi`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacGujarati`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacOriya`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacBengali`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacTamil`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacTelugu`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacKannada`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacMalayalam`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacSinhalese`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacBurmese`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacKhmer`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacThai`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacLaotian`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacGeorgian`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacArmenian`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacChineseSimp`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacTibetan`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacMongolian`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacEthiopic`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacCentralEurRoman`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacVietnamese`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacExtArabic`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacSymbol`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacDingbats`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacTurkish`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacCroatian`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacIcelandic`

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacRomanian`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacCeltic`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacGaelic`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacFarsi`
: Like MacArabic but uses Farsi digits

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacUkrainian`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacInuit`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacVT100`
: VT100102 font from Comm Toolbox: Latin-1 repertoire + box drawing etc

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacHFS`
: Meta-value, should never appear in a table

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin2`
: ISO 8859-2

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin3`
: ISO 8859-3

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin4`
: ISO 8859-4

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatinCyrillic`
ISO 8859-5

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatinArabic`
ISO 8859-6, =ASMO 708, =DOS CP 708

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatinGreek`
ISO 8859-7

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatinHebrew`
ISO 8859-8

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin5`
ISO 8859-9

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin6`
ISO 8859-10

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatinThai`
ISO 8859-11

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin7`
ISO 8859-13

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin8`
ISO 8859-14

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin9`
ISO 8859-15

Available in Mac OS X v10.0 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISOLatin10`
: ISO 8859-16

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSLatinUS`
: code page 437

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSGreek`
: code page 737 (formerly code page 437G)

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSBalticRim`
: code page 775

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSLatin1`
: code page 850, "Multilingual"

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSGreek1`
: code page 851

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSLatin2`
: code page 852, Slavic

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSCyrillic`
: code page 855, IBM Cyrillic

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSTurkish`
: code page 857, IBM Turkish

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSPortuguese`
: code page 860

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSIcelandic`
: code page 861

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSHebrew`
: code page 862

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSCanadianFrench`
: code page 863

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSArabic`
: code page 864

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSNordic`
: code page 865

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSRussian`
: code page 866

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSGreek2`
: code page 869, IBM Modern Greek

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSThai`
: code page 874, also for Windows

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSJapanese`
: code page 932, also for Windows

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSChineseSimplif`
: code page 936, also for Windows

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSKorean`
: code page 949, also for Windows; Unified Hangul Code

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingDOSChineseTrad`
: code page 950, also for Windows

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsLatin2`
: code page 1250, Central Europe

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsCyrillic`
: code page 1251, Slavic Cyrillic

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsGreek`
: code page 1253

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsLatin5`
: code page 1254, Turkish

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsHebrew`
: code page 1255

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsArabic`
: code page 1256

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsBalticRim`
: code page 1257

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsVietnamese`
: code page 1258

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingWindowsKoreanJohab`
: code page 1361, for Windows NT

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingANSEL`
: ANSEL (ANSI Z39.47)

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingJIS_X0201_76`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingJIS_X0208_83`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingJIS_X0208_90`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingJIS_X0212_90`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingJIS_C6226_78`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingShiftJIS_X0213_00`
: Shift-JIS format encoding of JIS X0213 planes 1 and 2

  Available in Mac OS X v10.2 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingShiftJIS_X0213_MenKuTen`
: JIS X0213 in plane-row-column notation

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingGB_2312_80`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingGBK_95`
: annex to GB 13000-93; for Windows 95

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingGB_18030_2000`
: Available in Mac OS X v10.2 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingKSC_5601_87`
: same as KSC 5601-92 without Johab annex

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingKSC_5601_92_Johab`
: KSC 5601-92 Johab annex

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingCNS_11643_92_P1`
: CNS 11643-1992 plane 1

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingCNS_11643_92_P2`
: CNS 11643-1992 plane 2

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingCNS_11643_92_P3`
: CNS 11643-1992 plane 3 (was plane 14 in 1986 version)

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_JP`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_JP_2`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_JP_1`
: RFC 2237

  Available in Mac OS X v10.2 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_JP_3`
: JIS X0213

  Available in Mac OS X v10.2 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_CN`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_CN_EXT`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingISO_2022_KR`
: Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEUC_JP`
: ISO 646, 1-byte katakana, JIS 208, JIS 212

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEUC_CN`
: ISO 646, GB 2312-80

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEUC_TW`
: ISO 646, CNS 11643-1992 Planes 1-16

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEUC_KR`
: ISO 646, KS C 5601-1987

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingShiftJIS`
: plain Shift-JIS

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingKOI8_R`
: Russian internet standard

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingBig5`
: Big-5 (has variants)

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingMacRomanLatin1`
: Mac OS Roman permuted to align with ISO Latin-1

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingHZ_GB_2312`
: HZ (RFC 1842, for Chinese mail & news)

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingBig5_HKSCS_1999`
: Big-5 with Hong Kong special char set supplement

  Available in Mac OS X v10.2 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingVISCII`
: RFC 1456, Vietnamese

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingKOI8_U`
: RFC 2319, Ukrainian

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingBig5_E`
: Taiwan Big-5E standard

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingNextStepJapanese`
: NextStep Japanese encoding

  Available in Mac OS X v10.4 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEBCDIC_US`
: basic EBCDIC-US

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingEBCDIC_CP037`
: code page 037, extended EBCDIC (Latin-1 set) for US,Canada...

  Available in Mac OS X v10.0 and later.

  Declared in `CFStringEncodingExt.h`.

Discussion
See the `CFStringEncodingExt.h` header file for the most current list of external string encodings and for more details.

Declared In
`CFStringEncodingExt.h`

Document Revision History

This table describes the changes to *CFString Reference*.

Date	Notes
2008-10-15	Added explanation of how locale argument affects CFStringCompareWithOptionsAndLocale and CFStringFindWithOptionsAndLocale functions.
2008-03-11	Added information to CFStringCreateExternalRepresentation function description about string encodings that do not include a BOM.
2007-10-31	Clarified the definition of the CFStringGetDoubleValue function.
2007-07-11	Updated to include new API in Mac OS X v10.5.
2007-07-10	Clarified encodings supported by C string representations.
2007-03-06	Clarified parameter descriptions for CFStringGetBytes; clarified behavior of NoCopy creation functions on failure.
2007-01-08	Corrected minor typographical errors.
2006-12-05	Clarified the return value of CFStringGetLength.
2006-06-28	Clarified the string argument to CFStringCreateWithCString.
2006-01-10	Clarified the meaning of kCFCompareAnchored.
2005-12-06	Made minor changes to text to conform to reference consistency guidelines.
2005-11-09	Corrected link in Companion Documents.
2005-04-29	Updated to include new API and encodings for Mac OS X version 10.4.
2004-11-02	Added note to Introduction regarding hash values.
2004-08-31	Added note regarding use of `-fconstant-cfstrings` with `CFSTR()`, and link to string formatting codes.
2004-04-22	Added note that specified ranges must not exceed length of string.
2004-02-21	Minor bug fix to description of the *`result`* parameter in `CFStringFindCharacterFromSet`.
2004-02-10	Minor bug fix related to `CFShowStr`.
2004-01-30	Minor bug fix related to Cocoa encoding conversion.

Document Revision History

Date	Notes
2003-08-01	Updated per new Mac OS X v10.3 API, and fixed other miscellaneous errors.
2003-01-01	First version of this document.

Index

B

Built-in String Encodings 61

C

CFShowStr function 12
CFSTR function 13
CFStringCompare function 13
CFStringCompareFlags data type 58
CFStringCompareWithOptions function 14
CFStringCompareWithOptionsAndLocale function 15
CFStringConvertEncodingToIANACharSetName function 16
CFStringConvertEncodingToNSStringEncoding function 16
CFStringConvertEncodingToWindowsCodepage function 17
CFStringConvertIANACharSetNameToEncoding function 17
CFStringConvertNSStringEncodingToEncoding function 18
CFStringConvertWindowsCodepageToEncoding function 18
CFStringCreateArrayBySeparatingStrings function 19
CFStringCreateArrayWithFindResults function 20
CFStringCreateByCombiningStrings function 21
CFStringCreateCopy function 21
CFStringCreateExternalRepresentation function 22
CFStringCreateFromExternalRepresentation function 23
CFStringCreateWithBytes function 24
CFStringCreateWithBytesNoCopy function 25
CFStringCreateWithCharacters function 26
CFStringCreateWithCharactersNoCopy function 27
CFStringCreateWithCString function 28
CFStringCreateWithCStringNoCopy function 29
CFStringCreateWithFileSystemRepresentation function 30
CFStringCreateWithFormat function 31
CFStringCreateWithFormatAndArguments function 32
CFStringCreateWithPascalString function 32
CFStringCreateWithPascalStringNoCopy function 33
CFStringCreateWithSubstring function 34
CFStringEncoding data type 58
CFStringEncodings data type 59
CFStringFind function 35
CFStringFindCharacterFromSet function 36
CFStringFindWithOptions function 37
CFStringFindWithOptionsAndLocale function 38
CFStringGetBytes function 39
CFStringGetCharacterAtIndex function 40
CFStringGetCharacterFromInlineBuffer function 41
CFStringGetCharacters function 41
CFStringGetCharactersPtr function 42
CFStringGetCString function 43
CFStringGetCStringPtr function 44
CFStringGetDoubleValue function 44
CFStringGetFastestEncoding function 45
CFStringGetFileSystemRepresentation function 45
CFStringGetIntValue function 46
CFStringGetLength function 47
CFStringGetLineBounds function 47
CFStringGetListOfAvailableEncodings function 48
CFStringGetMaximumSizeForEncoding function 49
CFStringGetMaximumSizeOfFileSystemRepresentation function 49
CFStringGetMostCompatibleMacStringEncoding function 50
CFStringGetNameOfEncoding function 50
CFStringGetParagraphBounds function 51
CFStringGetPascalString function 52
CFStringGetPascalStringPtr function 53

CFStringGetRangeOfComposedCharactersAtIndex function 53
CFStringGetSmallestEncoding function 54
CFStringGetSystemEncoding function 54
CFStringGetTypeID function 55
CFStringHasPrefix function 56
CFStringHasSuffix function 56
CFStringInitInlineBuffer function 57
CFStringInlineBuffer structure 59
CFStringIsEncodingAvailable function 57
CFStringRef data type 59

E

External String Encodings 64

I

Invalid String Encoding Flag 63

K

kCFCompareAnchored constant 60
kCFCompareBackwards constant 60
kCFCompareCaseInsensitive constant 60
kCFCompareDiacriticInsensitive constant 61
kCFCompareForcedOrdering constant 61
kCFCompareLocalized constant 61
kCFCompareNonliteral constant 60
kCFCompareNumerically constant 61
kCFCompareWidthInsensitive constant 61
kCFStringEncodingANSEL constant 75
kCFStringEncodingASCII constant 62
kCFStringEncodingBig5 constant 77
kCFStringEncodingBig5_E constant 78
kCFStringEncodingBig5_HKSCS_1999 constant 78
kCFStringEncodingCNS_11643_92_P1 constant 76
kCFStringEncodingCNS_11643_92_P2 constant 76
kCFStringEncodingCNS_11643_92_P3 constant 76
kCFStringEncodingDOSArabic constant 73
kCFStringEncodingDOSBalticRim constant 72
kCFStringEncodingDOSCanadianFrench constant 73
kCFStringEncodingDOSChineseSimplif constant 73
kCFStringEncodingDOSChineseTrad constant 74
kCFStringEncodingDOSCyrillic constant 72
kCFStringEncodingDOSGreek constant 72
kCFStringEncodingDOSGreek1 constant 72
kCFStringEncodingDOSGreek2 constant 73
kCFStringEncodingDOSHebrew constant 73
kCFStringEncodingDOSIcelandic constant 73
kCFStringEncodingDOSJapanese constant 73
kCFStringEncodingDOSKorean constant 74
kCFStringEncodingDOSLatin1 constant 72
kCFStringEncodingDOSLatin2 constant 72
kCFStringEncodingDOSLatinUS constant 72
kCFStringEncodingDOSNordic constant 73
kCFStringEncodingDOSPortuguese constant 72
kCFStringEncodingDOSRussian constant 73
kCFStringEncodingDOSThai constant 73
kCFStringEncodingDOSTurkish constant 72
kCFStringEncodingEBCDIC_CP037 constant 78
kCFStringEncodingEBCDIC_US constant 78
kCFStringEncodingEUC_CN constant 77
kCFStringEncodingEUC_JP constant 77
kCFStringEncodingEUC_KR constant 77
kCFStringEncodingEUC_TW constant 77
kCFStringEncodingGBK_95 constant 75
kCFStringEncodingGB_18030_2000 constant 75
kCFStringEncodingGB_2312_80 constant 75
kCFStringEncodingHZ_GB_2312 constant 77
kCFStringEncodingInvalidId constant 64
kCFStringEncodingISOLatin1 constant 62
kCFStringEncodingISOLatin10 constant 72
kCFStringEncodingISOLatin2 constant 70
kCFStringEncodingISOLatin3 constant 70
kCFStringEncodingISOLatin4 constant 70
kCFStringEncodingISOLatin5 constant 71
kCFStringEncodingISOLatin6 constant 71
kCFStringEncodingISOLatin7 constant 71
kCFStringEncodingISOLatin8 constant 71
kCFStringEncodingISOLatin9 constant 71
kCFStringEncodingISOLatinArabic constant 71
kCFStringEncodingISOLatinCyrillic constant 71
kCFStringEncodingISOLatinGreek constant 71
kCFStringEncodingISOLatinHebrew constant 71
kCFStringEncodingISOLatinThai constant 71
kCFStringEncodingISO_2022_CN constant 76
kCFStringEncodingISO_2022_CN_EXT constant 76
kCFStringEncodingISO_2022_JP constant 76
kCFStringEncodingISO_2022_JP_1 constant 76
kCFStringEncodingISO_2022_JP_2 constant 76
kCFStringEncodingISO_2022_JP_3 constant 76
kCFStringEncodingISO_2022_KR constant 77
kCFStringEncodingJIS_C6226_78 constant 75
kCFStringEncodingJIS_X0201_76 constant 75
kCFStringEncodingJIS_X0208_83 constant 75
kCFStringEncodingJIS_X0208_90 constant 75
kCFStringEncodingJIS_X0212_90 constant 75
kCFStringEncodingKOI8_R constant 77
kCFStringEncodingKOI8_U constant 78
kCFStringEncodingKSC_5601_87 constant 76
kCFStringEncodingKSC_5601_92_Johab constant 76

kCFStringEncodingMacArabic constant 67
kCFStringEncodingMacArmenian constant 69
kCFStringEncodingMacBengali constant 68
kCFStringEncodingMacBurmese constant 68
kCFStringEncodingMacCeltic constant 70
kCFStringEncodingMacCentralEurRoman constant 69
kCFStringEncodingMacChineseSimp constant 69
kCFStringEncodingMacChineseTrad constant 67
kCFStringEncodingMacCroatian constant 69
kCFStringEncodingMacCyrillic constant 67
kCFStringEncodingMacDevanagari constant 67
kCFStringEncodingMacDingbats constant 69
kCFStringEncodingMacEthiopic constant 69
kCFStringEncodingMacExtArabic constant 69
kCFStringEncodingMacFarsi constant 70
kCFStringEncodingMacGaelic constant 70
kCFStringEncodingMacGeorgian constant 68
kCFStringEncodingMacGreek constant 67
kCFStringEncodingMacGujarati constant 68
kCFStringEncodingMacGurmukhi constant 67
kCFStringEncodingMacHebrew constant 67
kCFStringEncodingMacHFS constant 70
kCFStringEncodingMacIcelandic constant 69
kCFStringEncodingMacInuit constant 70
kCFStringEncodingMacJapanese constant 67
kCFStringEncodingMacKannada constant 68
kCFStringEncodingMacKhmer constant 68
kCFStringEncodingMacKorean constant 67
kCFStringEncodingMacLaotian constant 68
kCFStringEncodingMacMalayalam constant 68
kCFStringEncodingMacMongolian constant 69
kCFStringEncodingMacOriya constant 68
kCFStringEncodingMacRoman constant 62
kCFStringEncodingMacRomanian constant 70
kCFStringEncodingMacRomanLatin1 constant 77
kCFStringEncodingMacSinhalese constant 68
kCFStringEncodingMacSymbol constant 69
kCFStringEncodingMacTamil constant 68
kCFStringEncodingMacTelugu constant 68
kCFStringEncodingMacThai constant 68
kCFStringEncodingMacTibetan constant 69
kCFStringEncodingMacTurkish constant 69
kCFStringEncodingMacUkrainian constant 70
kCFStringEncodingMacVietnamese constant 69
kCFStringEncodingMacVT100 constant 70
kCFStringEncodingNextStepJapanese constant 78
kCFStringEncodingNextStepLatin constant 62
kCFStringEncodingNonLossyASCII constant 63
kCFStringEncodingShiftJIS constant 77
kCFStringEncodingShiftJIS_X0213_00 constant 75
kCFStringEncodingShiftJIS_X0213_MenKuTen constant 75
kCFStringEncodingUnicode constant 62
kCFStringEncodingUTF16 constant 63
kCFStringEncodingUTF16BE constant 63
kCFStringEncodingUTF16LE constant 63
kCFStringEncodingUTF32 constant 63
kCFStringEncodingUTF32BE constant 63
kCFStringEncodingUTF32LE constant 63
kCFStringEncodingUTF8 constant 62
kCFStringEncodingVISCII constant 78
kCFStringEncodingWindowsArabic constant 74
kCFStringEncodingWindowsBalticRim constant 74
kCFStringEncodingWindowsCyrillic constant 74
kCFStringEncodingWindowsGreek constant 74
kCFStringEncodingWindowsHebrew constant 74
kCFStringEncodingWindowsKoreanJohab constant 75
kCFStringEncodingWindowsLatin1 constant 62
kCFStringEncodingWindowsLatin2 constant 74
kCFStringEncodingWindowsLatin5 constant 74
kCFStringEncodingWindowsVietnamese constant 74

S

String Comparison Flags 60