Core Foundation Framework Reference

2007-10-31

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

Simultaneously published in the United States and Canada.

Contents

Tables and Listings

Introduction

Framework	/System/Library/Frameworks/CoreFoundation.framework
Header file directories	/System/Library/Frameworks/CoreFoundation.framework/Headers
Declared in	CFArray.h CFAttributedString.h CFBag.h CFBase.h CFBinaryHeap.h CFBitVector.h CFBundle.h CFByteOrder.h CFCalendar.h CFCharacterSet.h CFData.h CFDate.h CFDateFormatter.h CFDictionary.h CFError.h CFFTPStream.h CFFileDescriptor.h CFHTTPStream.h CFHost.h CFLocale.h CFMachPort.h CFMessagePort.h CFNetServices.h CFNotificationCenter.h CFNumber.h CFNumberFormatter.h CFPlugIn.h CFPreferences.h CFPropertyList.h CFRunLoop.h CFSet.h CFSocket.h CFSocketStream.h CFStream.h CFString.h CFStringEncodingExt.h CFStringTokenizer.h CFTimeZone.h CFTree.h CFURL.h CFURLAccess.h CFUUID.h CFUserNotification.h

CFXMLNode.h
CFXMLParser.h

Core Foundation is a framework that provides fundamental software services useful to application services, application environments, and to applications themselves. Core Foundation also provides abstractions for common data types, facilitates internationalization with Unicode string storage, and offers a suite of utilities such as plug-in support, XML property lists, URL resource access, and preferences.

For a summary of new API introduced in Mac OS X v10.5, see *Core Foundation Reference Update*.

Opaque Types

CFAllocator Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBase.h
Companion guide	Memory Management Programming Guide for Core Foundation

Overview

CFAllocator is an opaque type that allocates and deallocates memory for you. You never have to allocate, reallocate, or deallocate memory directly for Core Foundation objects—and rarely should you. You pass CFAllocator objects into functions that create objects; these functions have "Create" embedded in their names, for example, `CFStringCreateWithPascalString`. The creation functions use the allocators to allocate memory for the objects they create.

Functions by Task

Creating an Allocator

`CFAllocatorCreate` (page 23)
Creates an allocator object.

Managing Memory with an Allocator

`CFAllocatorAllocate` (page 22)
Allocates memory using the specified allocator.

`CFAllocatorDeallocate` (page 23)
Deallocates a block of memory with a given allocator.

`CFAllocatorGetPreferredSizeForSize` (page 25)
Obtains the number of bytes likely to be allocated upon a specific request.

`CFAllocatorReallocate` (page 26)
Reallocates memory using the specified allocator.

Getting and Setting the Default Allocator

`CFAllocatorGetDefault` (page 25)
Gets the default allocator object for the current thread.

`CFAllocatorSetDefault` (page 27)
Sets the given allocator as the default for the current thread.

Getting an Allocator's Context

`CFAllocatorGetContext` (page 24)
Obtains the context of the specified allocator or of the default allocator.

Getting the CFAllocator Type ID

`CFAllocatorGetTypeID` (page 26)
Returns the type identifier for the CFAllocator opaque type.

Functions

CFAllocatorAllocate

Allocates memory using the specified allocator.

```
void * CFAllocatorAllocate (
   CFAllocatorRef allocator,
   CFIndex size,
   CFOptionFlags hint
);
```

Parameters

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

size
The size of the memory to allocate.

hint
A bitfield containing flags that suggest how memory is to be allocated. `0` indicates no hints. No hints are currently defined, so only `0` should be passed for this value.

Return Value
A pointer to the newly allocated memory.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
Dictionary
MoreSCF
String

Declared In
`CFBase.h`

CFAllocatorCreate

Creates an allocator object.

```
CFAllocatorRef CFAllocatorCreate (
   CFAllocatorRef allocator,
   CFAllocatorContext *context
);
```

Parameters

allocator

The existing allocator to use to allocate memory for the new allocator. Pass the `kCFAllocatorUseContext` (page 36) constant for this parameter to allocate memory using the appropriate function callback specified in the `context` parameter. Pass `NULL` or kCFAllocatorDefault (page 35) to allocate memory for the new allocator using the default allocator.

context

A structure of type `CFAllocatorContext` (page 33). The fields of this structure hold (among other things) function pointers to callbacks used for allocating, reallocating, and deallocating memory.

Return Value
The new allocator object, or `NULL` if there was a problem allocating memory. Ownership follows the Create Rule.

Discussion
You use this function to create custom allocators which you can then pass into various Core Foundation object-creation functions. You must implement a function callback that allocates memory and assign it to the `allocate` field of this structure. You typically also implement deallocate, reallocate, and preferred-size callbacks.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFBase.h`

CFAllocatorDeallocate

Deallocates a block of memory with a given allocator.

```
void CFAllocatorDeallocate (
   CFAllocatorRef allocator,
   void *ptr
);
```

Parameters

allocator

The allocator that was used to allocate the block of memory pointed to by `ptr`.

ptr

An untyped pointer to a block of memory to deallocate using `allocator`.

Discussion

If the allocator does not specify a `deallocate` callback function, the memory is not deallocated.

Special Considerations

You must use the same allocator to deallocate memory as was used to allocate it.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

Dictionary

MoreSCF

Declared In

`CFBase.h`

CFAllocatorGetContext

Obtains the context of the specified allocator or of the default allocator.

```
void CFAllocatorGetContext (
   CFAllocatorRef allocator,
   CFAllocatorContext *context
);
```

Parameters

allocator

The allocator to examine. Pass `NULL` to obtain the context of the default allocator.

context

On return, contains the context of *allocator*.

Discussion

An allocator's context, a structure of type `CFAllocatorContext`, holds pointers to various function callbacks (particularly those that allocate, reallocate, and deallocate memory for an object). The context also contains a version number and the `info` field for program-defined data. To obtain the value of the `info` field you usually first have to get an allocator's context.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In
CFBase.h

CFAllocatorGetDefault

Gets the default allocator object for the current thread.

```
CFAllocatorRef CFAllocatorGetDefault (
   void
);
```

Return Value
A reference to the default allocator for the current thread. If none has been explicitly set, returns the generic system allocator, `kCFAllocatorSystemDefault` (page 35). Ownership follows "The Get Rule" in *Memory Management Programming Guide for Core Foundation*.

Discussion
See the discussion for `CFAllocatorSetDefault` (page 27) for more detail on the default allocator and for advice on how and when to set a custom allocator as the default.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
MatrixMixerTest
String

Declared In
CFBase.h

CFAllocatorGetPreferredSizeForSize

Obtains the number of bytes likely to be allocated upon a specific request.

```
CFIndex CFAllocatorGetPreferredSizeForSize (
   CFAllocatorRef allocator,
   CFIndex size,
   CFOptionFlags hint
);
```

Parameters

allocator
The allocator to use, or `NULL` for the default allocator.

size
The number of bytes to allocate. If the value is `0` or less, the result is the same value.

hint
A bitfield of type `CFOptionsFlags`. Pass flags to the allocator that suggest how memory is to be allocated. `0` indicates no hints. No hints are currently defined, only `0` should be passed for this argument.

Return Value
The number of bytes likely to be allocated upon a specific request.

Discussion
The return value depends on the allocator's internal allocation strategy, and will be equal to or larger than `size`. Calling this function may help you better match your memory allocation or reallocation strategy to that of the allocator.

Note that the return value depends on the internal implementation of the allocator and the results may change from release to release or from platform to platform.

If no function callback is assigned to the `preferredSize` field of the allocator's context (see the `CFAllocatorContext` structure), then the value of `size` is returned.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFBase.h`

CFAllocatorGetTypeID

Returns the type identifier for the CFAllocator opaque type.

```
CFTypeID CFAllocatorGetTypeID (
   void
);
```

Return Value
The type identifier for the CFAllocator opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFBase.h`

CFAllocatorReallocate

Reallocates memory using the specified allocator.

```
void * CFAllocatorReallocate (
   CFAllocatorRef allocator,
   void *ptr,
   CFIndex newsize,
   CFOptionFlags hint
);
```

Parameters
`allocator`
The allocator to use for reallocating memory. Pass `NULL` to request the default allocator.

ptr

An untyped pointer to a block of memory to reallocate to a new size. If `ptr` is `NULL` and `newsize` is greater than 0, memory is allocated (using the `allocate` function callback of the allocator's context). If `ptr` is `NULL` and `newsize` is 0, the result is `NULL`.

newsize

The number of bytes to allocate. If you pass `0` and the `ptr` parameter is non-`NULL`, the block of memory that `ptr` points to is typically deallocated. If you pass 0 for this parameter and the `ptr` parameter is `NULL`, nothing happens and the result returned is `NULL`.

hint

A bitfield of type `CFOptionsFlags`. Pass flags to the allocator that suggest how memory is to be allocated. Zero indicates no hints. No hints are currently defined, only `0` should be passed for this argument.

Discussion

The `CFAllocatorReallocate` function's primary purpose is to reallocate a block of memory to a new (and usually larger) size. However, based on the values passed in certain of the parameters, this function can also allocate memory afresh or deallocate a given block of memory. The following summarizes the semantic combinations:

- If the `ptr` parameter is non- `NULL` and the `newsize` parameter is greater than `0`, the behavior is to reallocate.
- If the `ptr` parameter is `NULL` and the `newsize` parameter is greater than `0`, the behavior is to allocate.
- If the `ptr` parameter is non- `NULL` and the `newsize` parameter is `0`, the behavior is to deallocate.

The result of the `CFAllocatorReallocate` function is either an untyped pointer to a block of memory or `NULL`. A `NULL` result indicates either a failure to allocate memory or some other outcome, the precise interpretation of which is determined by the values of certain parameters and the presence or absence of callbacks in the allocator context. To summarize, a `NULL` result can mean one of the following:

- An error occurred in the attempt to allocate memory, such as insufficient free space.
- No `allocate`, `reallocate`, or `deallocate` function callback (depending on parameters) was defined in the allocator context.
- The semantic operation is "deallocate" (that is, there is no need to return anything).
- The `ptr` parameter is `NULL` and the requested size is 0.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFBase.h`

CFAllocatorSetDefault

Sets the given allocator as the default for the current thread.

```
void CFAllocatorSetDefault (
   CFAllocatorRef allocator
);
```

Parameters

allocator

The allocator to set as the default for the current thread.

Discussion

The `CFAllocatorSetDefault` function sets the allocator that is used in the current thread whenever `NULL` is specified as an allocator argument. Generally, most allocations use the default allocator. Because of this, the default allocator must be prepared to deal with arbitrary memory-allocation requests. In addition, the size and number of requests can change between releases.

A further characteristic of the default allocator is that it can never be released, even if another allocator replaces it as the default. Not only is it impractical to release a default allocator (because there might be caches created somewhere that refer to the allocator) but it is generally safer and more efficient to keep it around.

If you wish to use a custom allocator in a context, the best approach is to specify it in the first parameter of creation functions rather than to set it as the default. Generally, setting the default allocator is not encouraged. If you do set an allocator as the default, either do it for the life time of your application or do it in a nested fashion (that is, restore the previous allocator before you exit your context). The latter approach might be more appropriate for plug-ins or libraries that wish to set the default allocator.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFBase.h`

Callbacks

CFAllocatorAllocateCallBack

A prototype for a function callback that allocates memory of a requested size.

```
typedef void *(*CFAllocatorAllocateCallBack) (
    CFIndex allocSize,
    CFOptionFlags hint,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    CFIndex allocSize,
    CFOptionFlags hint,
    void *info
);
```

Parameters

allocSize

This function allocates a block of memory of at least `allocSize` bytes (always greater than 0).

hint

A bitfield that is currently not used (always set to 0).

info

An untyped pointer to program-defined data. Allocate memory for the data and assign a pointer to it. This data is often control information for the allocator. It may be `NULL`.

Return Value
A pointer to the start of the block.

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

Declared In
`CFBase.h`

CFAllocatorCopyDescriptionCallBack

A prototype for a function callback that provides a description of the specified data.

```
typedef CFStringRef (*CFAllocatorCopyDescriptionCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
    const void *info
);
```

Parameters

info

An untyped pointer to program-defined data.

Return Value
A CFString object that describes the allocator. The caller is responsible for releasing this object.

Discussion
A prototype for a function callback that provides a description of the data pointed to by the `info` field. In implementing this function, return a reference to a CFString object that describes your allocator, particularly some characteristics of your program-defined data.

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

Declared In
`CFBase.h`

CFAllocatorDeallocateCallBack

A prototype for a function callback that deallocates a block of memory.

```
typedef void (*CFAllocatorDeallocateCallBack) (
    void *ptr,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *ptr,
    void *info
);
```

Parameters

ptr

The block of memory to deallocate.

info

An untyped pointer to program-defined data.

Discussion

A prototype for a function callback that deallocates a given block of memory. In implementing this function, make the block of memory pointed to by `ptr` available for subsequent reuse by the allocator but unavailable for continued use by the program. The `ptr` parameter cannot be `NULL` and if the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBase.h`

CFAllocatorPreferredSizeCallBack

A prototype for a function callback that gives the size of memory likely to be allocated, given a certain request.

```
typedef CFIndex (*CFAllocatorPreferredSizeCallBack) (
    CFIndex size,
    CFOptionFlags hint,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFIndex MyCallBack (
    CFIndex size,
    CFOptionFlags hint,
    void *info
);
```

Parameters

size

The amount of memory requested.

hint

A bitfield that is currently not used (always set to 0).

info

An untyped pointer to program-defined data.

Return Value

The actual size the allocator is likely to allocate given this request.

Discussion

A prototype for a function callback that determines whether there is enough free memory to satisfy a request. In implementing this function, return the actual size the allocator is likely to allocate given a request for a block of memory of size `size`. The `hint` argument is a bitfield that you should currently not use.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBase.h`

CFAllocatorReallocateCallBack

A prototype for a function callback that reallocates memory of a requested size for an existing block of memory.

```
typedef void *(*CFAllocatorReallocateCallBack) (
    void *ptr,
    CFIndex newsize,
    CFOptionFlags hint,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    void *ptr,
    CFIndex newsize,
    CFOptionFlags hint,
    void *info
);
```

Parameters

ptr

The block of memory to resize.

newsize

The size of the new allocation.

hint

A bitfield that is currently not used (always set to 0).

info

An untyped pointer to program-defined data.

Return Value
Pointer to the new block of memory.

Discussion
In implementing this function, change the size of the block of memory pointed to by `ptr` to the size specified by `newsize` and return the pointer to the larger block of memory. Return `NULL` on any reallocation failure, leaving the old block of memory untouched. Also return `NULL` immediately if any of the following conditions if the ptr parameter is `NULL` or the *`newsize`* parameter is not greater than 0. Leave the contents of the old block of memory unchanged up to the lesser of the new or old sizes. If the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur. The `hint` argument is a bitfield that you should currently not use (that is, assign 0).

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

Declared In
`CFBase.h`

CFAllocatorReleaseCallBack

A prototype for a function callback that releases the given data.

```
typedef void (*CFAllocatorReleaseCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    const void *info
);
```

Parameters

`info`
The data to be released.

Discussion
A prototype for a function callback that releases the data pointed to by the `info` field. In implementing this function, release (or free) the data you have defined for the allocator context.

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

Declared In
`CFBase.h`

CFAllocatorRetainCallBack

A prototype for a function callback that retains the given data.

```
typedef const void *(*CFAllocatorRetainCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
    const void *info
);
```

Parameters

info

The data to be retained.

Discussion

A prototype for a function callback that retains the data pointed to by the `info` field. In implementing this function, retain the data you have defined for the allocator context in this field. (This might make sense only if the data is a Core Foundation object.)

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBase.h`

Data Types

CFAllocatorContext

A structure that defines the context or operating environment for an allocator (CFAllocator) object. Every Core Foundation allocator object must have a context defined for it.

```
struct CFAllocatorContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFAllocatorAllocateCallBack allocate;
    CFAllocatorReallocateCallBack reallocate;
    CFAllocatorDeallocateCallBack deallocate;
    CFAllocatorPreferredSizeCallBack preferredSize;
};
typedef struct CFAllocatorContext CFAllocatorContext;
```

Fields

`version`

An integer of type `CFIndex`. Assign the version number of the allocator. Currently the only valid value is 0.

`info`

An untyped pointer to program-defined data. Allocate memory for this data and assign a pointer to it. This data is often control information for the allocator. You may assign `NULL`.

`retain`

A prototype for a function callback that retains the data pointed to by the `info` field. In implementing this function, retain the data you have defined for the allocator context in this field. (This might make sense only if the data is a Core Foundation object.) You may set this function pointer to `NULL`.

`release`

A prototype for a function callback that releases the data pointed to by the `info` field. In implementing this function, release (or free) the data you have defined for the allocator context. You may set this function pointer to `NULL`, but doing so might result in memory leaks.

`copyDescription`

A prototype for a function callback that provides a description of the data pointed to by the `info` field. In implementing this function, return a reference to a CFString object that describes your allocator, particularly some characteristics of your program-defined data. You may set this function pointer to `NULL`, in which case Core Foundation will provide a rudimentary description.

`allocate`

A prototype for a function callback that allocates memory of a requested size. In implementing this function, allocate a block of memory of at least `size` bytes and return a pointer to the start of the block. The `hint` argument is a bitfield that you should currently not use (that is, assign 0). The `size` parameter should always be greater than 0. If it is not, or if problems in allocation occur, return `NULL`. This function pointer may not be assigned `NULL`.

`reallocate`

A prototype for a function callback that reallocates memory of a requested size for an existing block of memory. In implementing this function, change the size of the block of memory pointed to by `ptr` to the size specified by `newsize` and return the pointer to the larger block of memory. Return `NULL` on any reallocation failure, leaving the old block of memory untouched. Also return `NULL` immediately if any of the following conditions apply:

- The ptr parameter is `NULL`.
- The newsize parameter is not greater than 0.

  Leave the contents of the old block of memory unchanged up to the lesser of the new or old sizes. If the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur. The `hint` argument is a bitfield that you should currently not use (that is, assign 0). If you set this callback to `NULL` the `CFAllocatorReallocate` (page 26) function returns `NULL` in most cases when it attempts to use this allocator.

`deallocate`

A prototype for a function callback that deallocates a given block of memory. In implementing this function, make the block of memory pointed to by `ptr` available for subsequent reuse by the allocator but unavailable for continued use by the program. The `ptr` parameter cannot be `NULL` and if the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur. You can set this callback to `NULL`, in which case the `CFAllocatorDeallocate` (page 23) function has no effect.

`preferredSize`

A prototype for a function callback that determines whether there is enough free memory to satisfy a request. In implementing this function, return the actual size the allocator is likely to allocate given a request for a block of memory of size `size`. The `hint` argument is a bitfield that you should currently not use.

Discussion

See the "Memory Management" topic for information on creating a custom CFAllocator object and, as part of that procedure, the steps for creating a properly initialized `CFAllocatorContext` structure.

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

Declared In
`CFBase.h`

CFAllocatorRef

A reference to a CFAllocator object.

```
typedef const struct __CFAllocator *CFAllocatorRef;
```

Discussion
The `CFAllocatorRef` type is a reference type used in many Core Foundation parameters and function results. It refers to a CFAllocator object, which allocates, reallocates, and deallocates memory for Core Foundation objects.

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

Declared In
`CFBase.h`

Constants

Predefined Allocators

CFAllocator provides the following predefined allocators. In general, you should use `kCFAllocatorDefault` unless one of the special circumstances exist below.

```
const CFAllocatorRef kCFAllocatorDefault;
const CFAllocatorRef kCFAllocatorSystemDefault;
const CFAllocatorRef kCFAllocatorMalloc;
const CFAllocatorRef kCFAllocatorMallocZone;
const CFAllocatorRef kCFAllocatorNull;
const CFAllocatorRef kCFAllocatorUseContext;
```

Constants

`kCFAllocatorDefault`

This is a synonym for `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorSystemDefault`

Default system allocator.

You rarely need to use this.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorMalloc`
This allocator uses `malloc()`, `realloc()`, and `free()`.

Typically you should not use this allocator, use `kCFAllocatorDefault` instead. This allocator is useful as the `bytesDeallocator` in CFData or `contentsDeallocator` in CFString where the memory was obtained as a result of `malloc` type functions.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorMallocZone`
This allocator explicitly uses the default malloc zone, returned by `malloc_default_zone()`.

You should only use this when an object is safe to be allocated in non-scanned memory.

Available in Mac OS X v10.4 and later.

Declared in `CFBase.h`.

`kCFAllocatorNull`
This allocator does nothing—it allocates no memory.

This allocator is useful as the `bytesDeallocator` in CFData or `contentsDeallocator` in CFString where the memory should not be freed.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorUseContext`
Special allocator argument to `CFAllocatorCreate` (page 23)—it uses the functions given in the context to allocate the allocator.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

Declared In
`CFBase.h`

CFArray Reference

Derived From:	*CFPropertyList Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFArray.h
Companion guides	Collections Programming Topics for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFArray and its derived mutable type, *CFMutableArray Reference*, manage ordered collections of values called arrays. CFArray creates static arrays and CFMutableArray creates dynamic arrays.

You create a static array object using either the `CFArrayCreate` (page 41) or `CFArrayCreateCopy` (page 42) function. These functions return an array containing the values you pass in as arguments. (Note that arrays can't contain `NULL` pointers; in most cases, though, you can use the `kCFNull` (page 376) constant instead.) Values are not copied but retained using the retain callback provided when an array was created. Similarly, when a value is removed from an array, it is released using the release callback.

CFArray's two primitive functions `CFArrayGetCount` (page 43) and `CFArrayGetValueAtIndex` (page 46) provide the basis for all other functions in its interface. The `CFArrayGetCount` (page 43) function returns the number of elements in an array; `CFArrayGetValueAtIndex` (page 46) gives you access to an array's elements by index, with index values starting at 0.

A number of CFArray functions allow you to operate over a range of values in an array, for example `CFArrayApplyFunction` (page 39) lets you apply a function to values in an array, and `CFArrayBSearchValues` (page 39) searches an array for the value that matches its parameter. Recall that a range is defined as `{start, length}`, therefore to operate over the entire array the range you supply should be `{0, N}` (where `N` is the count of the array).

CFArray is "toll-free bridged" with its Cocoa Foundation counterpart, `NSArray`. 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 `NSArray *` parameter, you can pass in a `CFArrayRef`, and in a function where you see a `CFArrayRef` parameter, you can pass in an NSArray instance. This also applies to concrete subclasses of NSArray. See "Interchangeable Data Types" for more information on toll-free bridging.

Functions by Task

Creating an Array

`CFArrayCreate` (page 41)
Creates a new immutable array with the given values.

`CFArrayCreateCopy` (page 42)
Creates a new immutable array with the values from another array.

Examining an Array

`CFArrayBSearchValues` (page 39)
Searches an array for a value using a binary search algorithm.

`CFArrayContainsValue` (page 40)
Reports whether or not a value is in an array.

`CFArrayGetCount` (page 43)
Returns the number of values currently in an array.

`CFArrayGetCountOfValue` (page 43)
Counts the number of times a given value occurs in an array.

`CFArrayGetFirstIndexOfValue` (page 44)
Searches an array forward for a value.

`CFArrayGetLastIndexOfValue` (page 45)
Searches an array backward for a value.

`CFArrayGetValues` (page 46)
Fills a buffer with values from an array.

`CFArrayGetValueAtIndex` (page 46)
Retrieves a value at a given index.

Applying a Function to Elements

`CFArrayApplyFunction` (page 39)
Calls a function once for each element in range in an array.

Getting the CFArray Type ID

`CFArrayGetTypeID` (page 45)
Returns the type identifier for the CFArray opaque type.

Functions

CFArrayApplyFunction

Calls a function once for each element in range in an array.

```
void CFArrayApplyFunction (
   CFArrayRef theArray,
   CFRange range,
   CFArrayApplierFunction applier,
   void *context
);
```

Parameters

theArray

The array to whose elements to apply the function.

range

The range of values within *theArray* to which to apply the *applier* function. The range must not exceed the bounds of *theArray*. The range may be empty (length 0).

applier

The callback function to call once for each value in the given range in *theArray*. If there are values in the range that the *applier* function does not expect or cannot properly apply to, the behavior is undefined.

context

A pointer-sized program-defined value, which is passed as the second argument to the *applier* function, but is otherwise unused by this function. If the context is not what is expected by the applier function, the behavior is undefined.

Discussion

While this function iterates over a mutable collection, it is unsafe for the *applier* function to change the contents of the collection.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

GLUT

HID Manager Basics

HID Utilities Source

Son of Grab

Declared In

CFArray.h

CFArrayBSearchValues

Searches an array for a value using a binary search algorithm.

```
CFIndex CFArrayBSearchValues (
   CFArrayRef theArray,
   CFRange range,
   const void *value,
   CFComparatorFunction comparator,
   void *context
);
```

Parameters

theArray

An array, sorted from least to greatest according to the *comparator* function.

range

The range within *theArray* to search. The range must not exceed the bounds of *theArray*. The range may be empty (length 0).

value

The value for which to find a match in *theArray*. If *value*, or any other value in *theArray*, is not understood by the *comparator* callback, the behavior is undefined.

comparator

The function with the comparator function type signature that is used in the binary search operation to compare values in *theArray* with the given value. If there are values in the range that the *comparator* function does not expect or cannot properly compare, the behavior is undefined.

context

A pointer-sized program-defined value, which is passed as the third argument to the *comparator* function, but is otherwise unused by this function. If the context is not what is expected by the *comparator* function, the behavior is undefined.

Return Value

The return value is one of the following:

- The index of a value that matched, if the target value matches one or more in the range.
- Greater than or equal to the end point of the range, if the value is greater than all the values in the range.
- The index of the value greater than the target value, if the value lies between two of (or less than all of) the values in the range.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MoreSCF

Declared In

CFArray.h

CFArrayContainsValue

Reports whether or not a value is in an array.

```
Boolean CFArrayContainsValue (
   CFArrayRef theArray,
   CFRange range,
   const void *value
);
```

Parameters

theArray

The array to search.

range

The range within *theArray* to search. The range must not exceed the bounds of *theArray*). The range may be empty (length 0).

value

The value to match in *theArray*. The equal callback provided when *theArray* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theArray*, is not understood by the equal callback, the behavior is undefined.

Return Value

true, if *value* is in the specified range of *theArray*, otherwise false.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MoreSCF

PMPrinterPrintWithFile

SeeMyFriends

Declared In

CFArray.h

CFArrayCreate

Creates a new immutable array with the given values.

```
CFArrayRef CFArrayCreate (
   CFAllocatorRef allocator,
   const void **values,
   CFIndex numValues,
   const CFArrayCallBacks *callBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new array and its storage for values. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

values

A C array of the pointer-sized values to be in the new array. The values in the new array are ordered in the same order in which they appear in this C array. This value may be NULL if *numValues* is 0. This C array is not changed or freed by this function. If *values* is not a valid pointer to a C array of at least *numValues* elements, the behavior is undefined.

numValues

The number of values to copy from the values C array into the new array. This number will be the count of the new array—it must not be negative or greater than the number of elements in *values*.

callBacks

A pointer to a `CFArrayCallBacks` (page 50) structure initialized with the callbacks for the array to use on each value in the collection. The retain callback is used within this function, for example, to retain all of the new values from the values C array. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be `NULL`, which is treated as if a valid structure of version `0` with all fields `NULL` had been passed in. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this value is not a valid pointer to a `CFArrayCallBacks` (page 50) structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

If the collection contains only CFType objects, then pass a pointer to `kCFTypeArrayCallBacks` (page 52) (`&kCFTypeArrayCallBacks`) to use the default callback functions.

Return Value

A new immutable array containing *numValues* from *values*, or `NULL` if there was a problem creating the object. Ownership follows "The Create Rule" in *Memory Management Programming Guide for Core Foundation*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPrefTopScores

ColorSyncDevices-Cocoa

Dictionary

ImageClient

MoreSCF

Declared In

`CFArray.h`

CFArrayCreateCopy

Creates a new immutable array with the values from another array.

```
CFArrayRef CFArrayCreateCopy (
   CFAllocatorRef allocator,
   CFArrayRef theArray
);
```

Parameters

allocator

The allocator to use to allocate memory for the new array and its storage for values. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

theArray

The array to copy.

Return Value
A new CFArray object that contains the same values as *theArray*. Ownership follows "The Create Rule" in *Memory Management Programming Guide for Core Foundation*.

Discussion
The pointer values from *theArray* are copied into the new array; the values are also retained by the new array. The count of the new array is the same as *theArray*. The new array uses the same callbacks as *theArray*.

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

Related Sample Code
IdentitySample

Declared In
CFArray.h

CFArrayGetCount

Returns the number of values currently in an array.

```
CFIndex CFArrayGetCount (
   CFArrayRef theArray
);
```

Parameters

theArray
The array to examine.

Return Value
The number of values in *theArray*.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Dumper
HID Utilities
MoreSCF

Declared In
CFArray.h

CFArrayGetCountOfValue

Counts the number of times a given value occurs in an array.

```
CFIndex CFArrayGetCountOfValue (
   CFArrayRef theArray,
   CFRange range,
   const void *value
);
```

Parameters

theArray

The array to examine.

range

The range within *theArray* to search. The range must lie within the bounds of *theArray*). The range may be empty (length 0).

value

The value for which to find matches in *theArray*. The equal callback provided when *theArray* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theArray*, is not understood by the equal callback, the behavior is undefined.

Return Value

The number of times *value* occurs in *theArray*, within the specified range.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFArray.h

CFArrayGetFirstIndexOfValue

Searches an array forward for a value.

```
CFIndex CFArrayGetFirstIndexOfValue (
   CFArrayRef theArray,
   CFRange range,
   const void *value
);
```

Parameters

theArray

The array to examine.

range

The range within *theArray* to search. The range must lie within the bounds of *theArray*. The range may be empty (length 0). The search progresses from the lowest index defined by the range to the highest.

value

The value for which to find a match in *theArray*. The equal callback provided when *theArray* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theArray*, is not understood by the equal callback, the behavior is undefined.

Return Value

The lowest index of the matching values in the range, or -1 if no value in the range matched.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
ImageClient
MoreSCF

Declared In
`CFArray.h`

CFArrayGetLastIndexOfValue

Searches an array backward for a value.

```
CFIndex CFArrayGetLastIndexOfValue (
   CFArrayRef theArray,
   CFRange range,
   const void *value
);
```

Parameters

theArray
The array to examine.

range
The range within *theArray* to search. The range must not exceed the bounds of *theArray*. The range may be empty (length `0`). The search progresses from the highest index defined by the range to the lowest.

value
The value for which to find a match in *theArray*. The equal callback provided when *theArray* was created is used to compare. If the equal callback was `NULL`, pointer equality (in C, ==) is used. If value, or any other value in *theArray*, is not understood by the equal callback, the behavior is undefined.

Return Value
The highest index of the matching values in the range, or `-1` if no value in the range matched.

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

Declared In
`CFArray.h`

CFArrayGetTypeID

Returns the type identifier for the CFArray opaque type.

```
CFTypeID CFArrayGetTypeID (
   void
);
```

Return Value
The type identifier for the CFArray opaque type.

Special Considerations

CFMutableArray objects have the same type identifier as CFArray objects.

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

Related Sample Code
BetterAuthorizationSample
BSDLLCTest
FSMegaInfo
MoreSCF
RecentItems

Declared In
`CFArray.h`

CFArrayGetValueAtIndex

Retrieves a value at a given index.

```
const void * CFArrayGetValueAtIndex (
   CFArrayRef theArray,
   CFIndex idx
);
```

Parameters

`theArray`
The array to examine.

`idx`
The index of the value to retrieve. If the index is outside the index space of *`theArray`* (`0` to `N-1` inclusive (where `N` is the count of *`theArray`*), the behavior is undefined.

Return Value
The value at the *`idx`* index in *`theArray`*. If the return value is a Core Foundation Object, ownership follows "The Get Rule" in *Memory Management Programming Guide for Core Foundation*.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Dumper
HID Utilities
MoreSCF

Declared In
`CFArray.h`

CFArrayGetValues

Fills a buffer with values from an array.

```
void CFArrayGetValues (
   CFArrayRef theArray,
   CFRange range,
   const void **values
);
```

Parameters

theArray

The array to examine.

range

The range of values within *theArray* to retrieve. The range must lie within the bounds of *theArray*. The range may be empty (length 0), in which case no values are put into the buffer *values*.

values

A C array of pointer-sized values to be filled with values from *theArray*. The values in the C array are in the same order as they appear in *theArray*. If this value is not a valid pointer to a C array of at least `range.length` pointers, the behavior is undefined. If the values are Core Foundation objects, ownership follows "The Get Rule" in *Memory Management Programming Guide for Core Foundation*.

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

Related Sample Code
BSDLLCTest
MoreSCF

Declared In
`CFArray.h`

Callbacks

CFArrayApplierFunction

Prototype of a callback function that may be applied to every value in an array.

```
typedef void (*CFArrayApplierFunction) (
   const void *value,
   void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   const void *value,
   void *context
);
```

Parameters

value

The current value in an array.

context
The program-defined context parameter given to the applier function.

Discussion
This callback is passed to the `CFArrayApplyFunction` (page 39) function, which iterates over the values in an array and applies the behavior defined in the applier function to each value in an array.

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

Declared In
`CFArray.h`

CFArrayCopyDescriptionCallBack

Prototype of a callback function used to get a description of a value in an array.

```
typedef CFStringRef (*CFArrayCopyDescriptionCallBack) (
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
   const void *value
);
```

Parameters
value
The value to be described.

Return Value
A textual description of *value*. The caller is responsible for releasing this object.

Discussion
This callback is passed to `CFArrayCreate` (page 41) in a `CFArrayCallBacks` (page 50) structure. This callback is used by the `CFCopyDescription` (page 652) function.

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

Declared In
`CFArray.h`

CFArrayEqualCallBack

Prototype of a callback function used to determine if two values in an array are equal.

```
typedef Boolean (*CFArrayEqualCallBack) (
   const void *value1,
   const void *value2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
   const void *value1,
   const void *value2
);
```

Parameters

value1

A value in an array to be compared with *value2* for equality.

value2

A value in an array to be compared with *value1* for equality.

Return Value

`true` if *value1* and *value2* are equal, `false` otherwise.

Discussion

This callback is passed to `CFArrayCreate` (page 41) in a `CFArrayCallBacks` (page 50) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFArray.h`

CFArrayReleaseCallBack

Prototype of a callback function used to release a value before it's removed from an array.

```
typedef void (*CFArrayReleaseCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator

The array's allocator.

value

The value being removed from an array.

Discussion

This callback is passed to `CFArrayCreate` (page 41) in a `CFArrayCallBacks` (page 50) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFArray.h`

CFArrayRetainCallBack

Prototype of a callback function used to retain a value being added to an array.

```
typedef const void *(*CFArrayRetainCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator

The array's allocator.

value

The value being added to an array.

Return Value

The value to store in an array, which is usually the *value* parameter passed to this callback, but may be a different value if a different value should be stored in an array.

Discussion

This callback is passed to `CFArrayCreate` (page 41) in a `CFArrayCallBacks` (page 50) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFArray.h`

Data Types

CFArrayCallBacks

Structure containing the callbacks of a CFArray.

```
struct CFArrayCallBacks {
    CFIndex version;
    CFArrayRetainCallBack retain;
    CFArrayReleaseCallBack release;
    CFArrayCopyDescriptionCallBack copyDescription;
    CFArrayEqualCallBack equal;
};
typedef struct CFArrayCallBacks CFArrayCallBacks;
```

Fields

`version`

The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is 0.

`retain`

The callback used to retain each value as they are added to the collection. If `NULL`, values are not retained. See `CFArrayRetainCallBack` (page 50) for a description of this callback.

`release`

The callback used to release values as they are removed from the collection. If `NULL`, values are not released. See `CFArrayReleaseCallBack` (page 49) for a description of this callback.

`copyDescription`

The callback used to create a descriptive string representation of each value in the collection. If `NULL`, the collection will create a simple description of each value. See `CFArrayCopyDescriptionCallBack` (page 48) for a description of this callback.

`equal`

The callback used to compare values in the array for equality for some operations. If `NULL`, the collection will use pointer equality to compare values in the collection. See `CFArrayEqualCallBack` (page 48) for a description of this callback.

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

Declared In
`CFArray.h`

CFArrayRef

A reference to an immutable array object.

```
typedef const struct __CFArray *CFArrayRef;
```

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

Declared In
`CFArray.h`

Constants

Predefined Callback Structures

CFArray provides a predefined callback structure appropriate for use when the values in a CFArray are all CFType-derived objects.

```
const CFArrayCallBacks kCFTypeArrayCallBacks;
```

Constants

`kCFTypeArrayCallBacks`

Predefined `CFArrayCallBacks` (page 50) structure containing a set of callbacks appropriate for use when the values in a CFArray are all CFType-derived objects. The retain callback is `CFRetain`, the release callback is `CFRelease`, the copy callback is `CFCopyDescription`, and the equal callback is `CFEqual`. Therefore, if you use this constant when creating the collection, items are automatically retained when added to the collection, and released when removed from the collection.

Available in Mac OS X v10.0 and later.

Declared in `CFArray.h`.

CFAttributedString Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFAttributedString.h CFBase.h
Companion guides	Property List Programming Topics for Core Foundation String Programming Guide for Core Foundation Data Formatting Guide for Core Foundation

Overview

Instances of CFAttributedString manage character strings and associated sets of attributes (for example, font and kerning information) that apply to individual characters or ranges of characters in the string. CFAttributedString as defined in CoreFoundation provides the basic container functionality, while higher levels provide definitions for standard attributes, their values, and additional behaviors involving these. CFAttributedString represents an immutable string—use CFMutableAttributedString to create and manage an attributed string that can be changed after it has been created.

iOS Note: While Core Foundation on iOS contains CFAttributedString, there are no additions to the APIs in UIKit to add specific attributes such as font, style, or color, and there are no APIs to draw attributed strings.

CFAttributedString is not a "subclass" of CFString; that is, it does not respond to CFString function calls. CFAttributedString conceptually contains a CFString to which it applies attributes. This protects you from ambiguities caused by the semantic differences between simple and attributed string.

Attributes are identified by key/value pairs stored in CFDictionary objects. Keys must be CFString objects, while the corresponding values are CFType objects of an appropriate type. See the attribute constants in *NSAttributedString Application Kit Additions Reference* for standard attribute names.

Important: Attribute dictionaries set for an attributed string must always be created with kCFCopyStringDictionaryKeyCallbacks for their dictionary key callbacks and kCFTypeDictionaryValueCallBacks for their value callbacks; otherwise it's an error.

On Mac OS X, CFAttributedString is "toll-free bridged" with its Cocoa Foundation counterpart, `NSAttributedString`. 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 `NSAttributedString *` parameter, you can pass in a `CFAttributedStringRef`, and in a function where you see a

`CFAttributedStringRef` parameter, you can pass in an `NSAttributedString` instance. This also applies to concrete subclasses of `NSAttributedString`. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a CFAttributedString

`CFAttributedStringCreate` (page 54)
Creates an attributed string with specified string and attributes.

`CFAttributedStringCreateCopy` (page 55)
Creates an immutable copy of an attributed string.

`CFAttributedStringCreateWithSubstring` (page 56)
Creates a sub-attributed string from the specified range.

`CFAttributedStringGetLength` (page 59)
Returns the length of the attributed string in characters.

`CFAttributedStringGetString` (page 60)
Returns the string for an attributed string.

Accessing Attributes

`CFAttributedStringGetAttribute` (page 56)
Returns the value of a given attribute of an attributed string at a specified location.

`CFAttributedStringGetAttributes` (page 58)
Returns the attributes of an attributed string at a specified location.

`CFAttributedStringGetAttributeAndLongestEffectiveRange` (page 57)
Returns the value of a given attribute of an attributed string at a specified location.

`CFAttributedStringGetAttributesAndLongestEffectiveRange` (page 58)
Returns the attributes of an attributed string at a specified location.

Getting Attributed String Properties

`CFAttributedStringGetTypeID` (page 60)
Returns the type identifier for the CFAttributedString opaque type.

Functions

CFAttributedStringCreate

Creates an attributed string with specified string and attributes.

```
CFAttributedStringRef CFAttributedStringCreate (
   CFAllocatorRef alloc,
   CFStringRef str,
   CFDictionaryRef attributes
);
```

Parameters

alloc

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

str

A string that specifies the characters to use in the new attributed string. This value is copied.

attributes

A dictionary that contains the attributes to apply to the new attributed string. This value is copied.

Return Value

An attributed string that contains the characters from *str* and the attributes specified by *attributes*. The result is `NULL` if there was a problem in creating the attributed string. Ownership follows the Create Rule.

Discussion

Note that both the string and the attributes dictionary are copied. The specified attributes are applied to the whole string. If you want to apply different attributes to different ranges of the string, you should use a mutable attributed string.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

CoreTextTest

Declared In

`CFAttributedString.h`

CFAttributedStringCreateCopy

Creates an immutable copy of an attributed string.

```
CFAttributedStringRef CFAttributedStringCreateCopy (
   CFAllocatorRef alloc,
   CFAttributedStringRef aStr
);
```

Parameters

alloc

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

aStr

The attributed string to copy.

Return Value

An immutable attributed string with characters and attributes identical to those of *aStr*. Returns `NULL` if there was a problem copying the object. Ownership follows the Create Rule.

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

Declared In
`CFAttributedString.h`

CFAttributedStringCreateWithSubstring

Creates a sub-attributed string from the specified range.

```
CFAttributedStringRef CFAttributedStringCreateWithSubstring (
   CFAllocatorRef alloc,
   CFAttributedStringRef aStr,
   CFRange range
);
```

Parameters

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

theString
The attributed string to copy.

range
The range of the attributed string to copy. *range* must not exceed the bounds of *aStr*.

Return Value
A new attributed string whose string and attributes are copied from from the specified range of the supplied attributed string. Returns `NULL` if there was a problem copying the object. Ownership follows the Create Rule.

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

Declared In
`CFAttributedString.h`

CFAttributedStringGetAttribute

Returns the value of a given attribute of an attributed string at a specified location.

```
CFTypeRef CFAttributedStringGetAttribute (
   CFAttributedStringRef aStr,
   CFIndex loc,
   CFStringRef attrName,
   CFRange *effectiveRange
);
```

Parameters

str
The attributed string to examine.

loc
The location in *str* at which to determine the attributes. *loc* must not exceed the bounds of *str*.

attrName
The name of the attribute whose value you want to determine.

effectiveRange
If not NULL, upon return contains a range including *loc* over which exactly the same set of attributes apply as at *loc*.

Return Value
The value of the specified attribute at the specified location in *str*. Ownership follows the Get Rule.

Discussion
For performance reasons, a range returned in *effectiveRange* is not necessarily the maximal range. If you need the maximum range, you should use CFAttributedStringGetAttributeAndLongestEffectiveRange (page 57).

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

Declared In
CFAttributedString.h

CFAttributedStringGetAttributeAndLongestEffectiveRange

Returns the value of a given attribute of an attributed string at a specified location.

```
CFTypeRef CFAttributedStringGetAttributeAndLongestEffectiveRange (
   CFAttributedStringRef aStr,
   CFIndex loc,
   CFStringRef attrName,
   CFRange inRange,
   CFRange *longestEffectiveRange
);
```

Parameters

str
The attributed string to examine.

loc
The location in *str* at which to determine the attributes. It is a programming error for *loc* to specify a location outside the bounds of *str*.

attrName
The name of the attribute whose value you want to determine.

inRange
The range in *str* within which you want to find the longest effective range of the attributes at *loc*. *inRange* must not exceed the bounds of *str*.

effectiveRange
If not NULL, upon return contains the maximal range within *inRange* over which the exact same set of attributes apply. The returned range is clipped to *inRange*.

Return Value
A dictionary that contains the attributes of *str* at the specified location. Ownership follows the Get Rule.

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

Declared In
CFAttributedString.h

CFAttributedStringGetAttributes

Returns the attributes of an attributed string at a specified location.

```
CFDictionaryRef CFAttributedStringGetAttributes (
   CFAttributedStringRef aStr,
   CFIndex loc,
   CFRange *effectiveRange
);
```

Parameters

str

The attributed string to examine.

loc

The location in *str* at which to determine the attributes. *loc* must not exceed the bounds of *str*.

effectiveRange

If not NULL, upon return contains a range including *loc* over which exactly the same set of attributes apply as at *loc*.

Return Value
A dictionary that contains the attributes of *str* at the specified location. Ownership follows the Get Rule.

Discussion
For performance reasons, a range returned in *effectiveRange* is not necessarily the maximal range. If you need the maximum range, you should use CFAttributedStringGetAttributesAndLongestEffectiveRange (page 58).

Note that the returned attribute dictionary might change in unpredictable ways if the attributed string is edited after this call. If you want to preserve the state of the dictionary, you should make an actual copy of it rather than just retaining it. In addition, you should make no assumptions about the relationship of the actual dictionary returned by this call and the dictionary originally used to set the attributes, other than the fact that the values stored in the dictionaries will be identical (that is, ==) to those originally specified.

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

Declared In
CFAttributedString.h

CFAttributedStringGetAttributesAndLongestEffectiveRange

Returns the attributes of an attributed string at a specified location.

```
CFDictionaryRef CFAttributedStringGetAttributesAndLongestEffectiveRange (
   CFAttributedStringRef aStr,
   CFIndex loc,
   CFRange inRange,
   CFRange *longestEffectiveRange
);
```

Parameters

str

The attributed string to examine.

loc

The location in *str* at which to determine the attributes. *loc* must not exceed the bounds of *str*.

inRange

The range in *str* within to find the longest effective range of the attributes at *loc*. *inRange* must not exceed the bounds of *str*.

effectiveRange

If not `NULL`, upon return contains the maximal range within *inRange* over which the exact same set of attributes apply. The returned range is clipped to *inRange*.

Return Value

A dictionary that contains the attributes of *str* at the specified location. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFAttributedString.h`

CFAttributedStringGetLength

Returns the length of the attributed string in characters.

```
CFIndex CFAttributedStringGetLength (
   CFAttributedStringRef aStr
);
```

Parameters

str

The attributed string to examine.

Return Value

The length of the attributed string in characters; this is the same as `CFStringGetLength(CFAttributedStringGetString(aStr))`.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

CoreTextTest

Declared In

`CFAttributedString.h`

CFAttributedStringGetString

Returns the string for an attributed string.

```
CFStringRef CFAttributedStringGetString (
   CFAttributedStringRef aStr
);
```

Parameters

aStr

The attributed string to examine.

Return Value

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

Discussion

For performance reasons, the string returned will often be the backing store of the attributed string, and it might therefore change if the attributed string is edited. However, this is an implementation detail, and you should not rely on this behavior.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFAttributedString.h`

CFAttributedStringGetTypeID

Returns the type identifier for the CFAttributedString opaque type.

```
CFTypeID CFAttributedStringGetTypeID (
   void
);
```

Return Value

The type identifier for the CFAttributedString opaque type.

Discussion

CFMutableAttributedString objects have the same type identifier as CFAttributedString objects.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFAttributedString.h`

Data Types

CFAttributedStringRef

A reference to a CFAttributedString object.

```
typedef const struct __CFAttributedString *CFAttributedStringRef;
```

Discussion
The `CFAttributedStringRef` type refers to an object that combines a CFString object with a collection of attributes that specify how the characters in the string should be displayed. CFAttributedString is an opaque type that defines the characteristics and behavior of CFAttributedString objects.

Values of type `CFAttributedStringRef` may refer to immutable or mutable strings, as CFMutableAttributedString objects respond to all functions intended for immutable CFAttributedString objects. Functions which accept `CFAttributedStringRef` values, and which need to hold on to the values immutably, should call `CFAttributedStringCreateWithSubstring` (page 56) (instead of `CFRetain` (page 657)) to do so.

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

Declared In
`CFAttributedString.h`

CFBag Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBag.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFBag and its derived mutable type, CFMutableBag, manage non-sequential collections of values called bags in which there can be duplicate values. CFBag creates static bags and CFMutableBag creates dynamic bags.

Use bags or sets as an alternative to arrays when the order of elements isn't important and performance in testing whether a value is contained in the collection is a consideration—while arrays are ordered, testing for membership is slower than with bags or sets. Use bags over sets if you want to allow duplicate values in your collections.

You create a static bag object using either the `CFBagCreate` (page 65) or `CFBagCreateCopy` (page 66) function. These functions return a bag containing the values you pass in as arguments. (Note that bags can't contain `NULL` pointers; in most cases, though, you can use the `kCFNull` constant instead.) Values are not copied but retained using the retain callback provided when the bag was created. Similarly, when a value is removed from a bag, it is released using the release callback.

CFBag provides functions for querying the values of a bag. The `CFBagGetCount` (page 67) returns the number of values in a bag, the `CFBagContainsValue` (page 65) function checks if a value is in a bag, and `CFBagGetValues` (page 69) returns a C array containing all the values in a bag.

The `CFBagApplyFunction` (page 64) function lets you apply a function to all values in a bag.

Functions by Task

Creating a Bag

`CFBagCreate` (page 65)
Creates an immutable bag containing specified values.

`CFBagCreateCopy` (page 66)
Creates an immutable bag with the values of another bag.

Examining a Bag

`CFBagContainsValue` (page 65)
Reports whether or not a value is in a bag.

`CFBagGetCount` (page 67)
Returns the number of values currently in a bag.

`CFBagGetCountOfValue` (page 67)
Returns the number of times a value occurs in a bag.

`CFBagGetValue` (page 68)
Returns a requested value from a bag.

`CFBagGetValueIfPresent` (page 69)
Reports whether or not a value is in a bag, and returns that value indirectly if it exists.

`CFBagGetValues` (page 69)
Fills a buffer with values from a bag.

Applying a Function to the Contents of a Bag

`CFBagApplyFunction` (page 64)
Calls a function once for each value in a bag.

Getting the CFBag Type ID

`CFBagGetTypeID` (page 68)
Returns the type identifier for the CFBag opaque type.

Functions

CFBagApplyFunction

Calls a function once for each value in a bag.

```
void CFBagApplyFunction (
   CFBagRef theBag,
   CFBagApplierFunction applier,
   void *context
);
```

Parameters

theBag
The bag to operate upon.

applier
The callback function to call once for each value in the *theBag*. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. If there are values in the range that the *applier* function does not expect or cannot properly apply to, the behavior is undefined.

context

A pointer-sized program-defined value, which is passed as the second parameter to the *applier* function, but is otherwise unused by this function. If the context is not what is expected by the applier function, the behavior is undefined.

Discussion
While this function iterates over a mutable collection, it is unsafe for the *applier* function to change the contents of the collection.

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

Declared In
CFBag.h

CFBagContainsValue

Reports whether or not a value is in a bag.

```
Boolean CFBagContainsValue (
   CFBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag to examine.

value

The value to match in *theBag*. The equal callback provided when *theBag* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theBag*, is not understood by the equal callback, the behavior is undefined.

Return Value
true if *value* is contained in *theBag*, otherwise false.

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

Declared In
CFBag.h

CFBagCreate

Creates an immutable bag containing specified values.

```
CFBagRef CFBagCreate (
   CFAllocatorRef allocator,
   const void **values,
   CFIndex numValues,
   const CFBagCallBacks *callBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

values

A C array of the pointer-sized values to be in the new bag. This parameter may be `NULL` if the *numValues* parameter is 0. The C array is not changed or freed by this function. `values` must be a valid pointer to a C array of at least `numValues` elements.

numValues

The number of values to copy from the `values` C array in the new CFBag object. If the number is negative or is greater than the actual number of values, the behavior is undefined.

callBacks

A pointer to a CFBagCallBacks (page 74) structure initialized with the callbacks to use to retain, release, describe, and compare values in the bag. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations. This parameter may be `NULL`, which is treated as if a valid structure of version 0 with all fields `NULL` had been passed in. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a CFBagCallBacks (page 74) structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined. If the collection contains CFType objects only, then pass `kCFTypeBagCallBacks` (page 75) as this parameter to use the default callback functions.

Return Value

A new bag, 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.

Declared In

`CFBag.h`

CFBagCreateCopy

Creates an immutable bag with the values of another bag.

```
CFBagRef CFBagCreateCopy (
   CFAllocatorRef allocator,
   CFBagRef theBag
);
```

Parameters

allocator

The allocator to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

theBag

The bag to copy. The pointer values from *theBag* are copied into the new bag. However, the values are also retained by the new bag. The count of the new bag is the same as the count of *theBag*. The new bag uses the same callbacks as *theBag*.

Return Value
A new bag that contains the same values as *theBag*, 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.

Declared In
CFBag.h

CFBagGetCount

Returns the number of values currently in a bag.

```
CFIndex CFBagGetCount (
   CFBagRef theBag
);
```

Parameters

theBag

The bag to examine.

Return Value
The number of values in *theBag*.

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

Declared In
CFBag.h

CFBagGetCountOfValue

Returns the number of times a value occurs in a bag.

```
CFIndex CFBagGetCountOfValue (
   CFBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag to examine.

value

The value for which to find matches in *theBag*. The equal callback provided when *theBag* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theBag*, is not understood by the equal callback, the behavior is undefined.

Return Value
The number of times *value* occurs in *theBag*.

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

Declared In
CFBag.h

CFBagGetTypeID

Returns the type identifier for the CFBag opaque type.

```
CFTypeID CFBagGetTypeID (
   void
);
```

Return Value
The type identifier for the CFBag opaque type.

Special Considerations

CFMutableBag objects have the same type identifier as CFBag objects.

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

Declared In
CFBag.h

CFBagGetValue

Returns a requested value from a bag.

```
const void * CFBagGetValue (
   CFBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag to examine.

value

The value for which to find matches in *theBag*. The equal callback provided when *theBag* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theBag*, is not understood by the equal callback, the behavior is undefined.

Return Value
A pointer to *value*, or NULL if *value* is not in *theBag*. If the value is a Core Foundation object, ownership follows the Get Rule.

Discussion
Depending on the implementation of the equal callback specified when creating *theBag*, the value returned may not have the same pointer equality as *value*.

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

Declared In
`CFBag.h`

CFBagGetValueIfPresent

Reports whether or not a value is in a bag, and returns that value indirectly if it exists.

```
Boolean CFBagGetValueIfPresent (
   CFBagRef theBag,
   const void *candidate,
   const void **value
);
```

Parameters

theBag
The bag to be searched.

candidate
The value for which to find matches in *theBag*. The equal callback provided when *theBag* was created is used to compare. If the equal callback was `NULL`, pointer equality (in C, ==) is used. If *candidate*, or any other value in *theBag*, is not understood by the equal callback, the behavior is undefined.

value
A pointer to a value object. Set to the matching value if it exists in the bag, otherwise `NULL`. If the value is a Core Foundation object, ownership follows the Get Rule.

Return Value
`true` if *value* is present in *theBag*, otherwise `false`.

Discussion
Depending on the implementation of the equal callback specified when creating *theBag*, the value returned in *value* may not have the same pointer equality as *candidate*.

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

Declared In
`CFBag.h`

CFBagGetValues

Fills a buffer with values from a bag.

```
void CFBagGetValues (
   CFBagRef theBag,
   const void **values
);
```

Parameters

theBag
The bag to examine.

values
A C array of pointer-sized values to be filled with values from *theBag*. The value must be a valid C array of the appropriate type and size (that is, a size equal to the count of *theBag*).

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

Declared In
CFBag.h

Callbacks

CFBagApplierFunction

Prototype of a callback function that may be applied to every value in a bag.

```
typedef void (*CFBagApplierFunction) (
   const void *value,
   void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   const void *value,
   void *context
);
```

Parameters

value
The current value in a bag.

context
The program-defined context parameter given to the apply function.

Discussion
This callback is passed to the `CFBagApplyFunction` (page 64) function which iterates over the values in a bag and applies the behavior defined in the applier function to each value in a bag.

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

Declared In
CFBag.h

CFBagCopyDescriptionCallBack

Prototype of a callback function used to get a description of a value in a bag.

```
typedef CFStringRef (*CFBagCopyDescriptionCallBack) (
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
   const void *value
);
```

Parameters

value

The value to be described.

Return Value

A textual description of *value*. mmancreate

Discussion

This callback is passed to `CFBagCreate` (page 65) in a `CFBagCallBacks` (page 74) structure. This callback is used by the `CFCopyDescription` (page 652) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBag.h`

CFBagEqualCallBack

Prototype of a callback function used to determine if two values in a bag are equal.

```
typedef Boolean (*CFBagEqualCallBack) (
   const void *value1,
   const void *value2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
   const void *value1,
   const void *value2
);
```

Parameters

value1

A value in the bag.

value2

Another value in the bag.

Return Value

`true` if *value1* and *value2* are equal, `false` otherwise.

Discussion

This callback is passed to `CFBagCreate` (page 65) in a `CFBagCallBacks` (page 74) structure.

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

Declared In
`CFBag.h`

CFBagHashCallBack

Prototype of a callback function invoked to compute a hash code for a value. Hash codes are used when values are accessed, added, or removed from a collection.

```
typedef CFHashCode      (*CFBagHashCallBack) (
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFHashCode CFBagHashCallBack (
   const void *value
);
```

Parameters

value

The value used to compute the hash code.

Return Value
An integer that can be used as a table address in a hash table structure.

Discussion
This callback is passed to `CFBagCreate` (page 65) in a `CFBagCallBacks` (page 74) structure.

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

Declared In
`CFBag.h`

CFBagReleaseCallBack

Prototype of a callback function used to release a value before it's removed from a bag.

```
typedef void (*CFBagReleaseCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator

The bag's allocator.

value

The value being removed from the bag.

Discussion

This callback is passed to `CFBagCreate` (page 65) in a `CFBagCallBacks` (page 74) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBag.h`

CFBagRetainCallBack

Prototype of a callback function used to retain a value being added to a bag.

```
typedef const void *(*CFBagRetainCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator

The bag's allocator.

value

The value being added to the bag.

Return Value

The value to store in the bag, which is usually the *value* parameter passed to this callback, but may be a different value if a different value should be stored in the collection.

Discussion

This callback is passed to `CFBagCreate` (page 65) in a `CFBagCallBacks` (page 74) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBag.h`

Data Types

CFBagCallBacks

This structure contains the callbacks used to retain, release, describe, and compare the values of a CFBag object.

```
struct CFBagCallBacks {
    CFIndex version;
    CFBagRetainCallBack retain;
    CFBagReleaseCallBack release;
    CFBagCopyDescriptionCallBack copyDescription;
    CFBagEqualCallBack equal;
    CFBagHashCallBack hash;
};
typedef struct CFBagCallBacks CFBagCallBacks;
```

Fields

`version`

The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is 0.

`retain`

The callback used to retain each value as they are added to the collection. If `NULL`, values are not retained. See `CFBagRetainCallBack` (page 73) for a descriptions of this function's parameters.

`release`

The callback used to release values as they are removed from the collection. If `NULL`, values are not released. See `CFBagReleaseCallBack` (page 72) for a description of this callback.

`copyDescription`

The callback used to create a descriptive string representation of each value in the collection. If `NULL`, the collection will create a simple description of each value. See `CFBagCopyDescriptionCallBack` (page 70) for a description of this callback.

`equal`

The callback used to compare values in the collection for equality for some operations. If `NULL`, the collection will use pointer equality to compare values in the collection. See `CFBagEqualCallBack` (page 71) for a description of this callback.

`hash`

The callback used to compute a hash code for values in a collection. If `NULL`, the collection computes a hash code by converting the pointer value to an integer. See `CFBagHashCallBack` (page 72) for a description of this callback.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBag.h`

CFBagRef

A reference to an immutable bag object.

```
typedef const struct __CFBag *CFBagRef;
```

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

Declared In
`CFBag.h`

Constants

Predefined Callback Structures

CFBag provides some predefined callbacks for your convenience.

```
const CFBagCallBacks kCFTypeBagCallBacks;
const CFBagCallBacks kCFCopyStringBagCallBacks;
```

Constants

`kCFTypeBagCallBacks`
: Predefined `CFBagCallBacks` (page 74) structure containing a set of callbacks appropriate for use when the values in a CFBag are all CFType-derived objects. The retain callback is `CFRetain`, the release callback is `CFRelease`, the copy callback is `CFCopyDescription`, the equal callback is `CFEqual`, and the hash callback is `CFHash`. Therefore, if you use this constant when creating the collection, items are automatically retained when added to the collection, and released when removed from the collection.

  Available in Mac OS X v10.0 and later.

  Declared in `CFBag.h`.

`kCFCopyStringBagCallBacks`
: Predefined `CFBagCallBacks` (page 74) structure containing a set of callbacks appropriate for use when the values in a CFBag are all CFString objects. The bag makes immutable copies of the strings placed into it.

  Available in Mac OS X v10.0 and later.

  Declared in `CFBag.h`.

CFBinaryHeap Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBinaryHeap.h
Companion guide	Collections Programming Topics

Overview

`CFBinaryHeap` implements a container that stores values sorted using a binary search algorithm. All binary heaps are mutable; there is not a separate immutable variety. Binary heaps can be useful as priority queues.

Functions

CFBinaryHeapAddValue

Adds a value to a binary heap.

```
void CFBinaryHeapAddValue (
   CFBinaryHeapRef heap,
   const void *value
);
```

Parameters

heap

The binary heap to use.

value

The value to add to the binary heap. The value is retained by the binary heap using the retain callback provided in the `CFBinaryHeapCallBacks` (page 86) structure when the binary heap was created.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapApplyFunction

Iteratively applies a function to all the values in a binary heap.

```
void CFBinaryHeapApplyFunction (
   CFBinaryHeapRef heap,
   CFBinaryHeapApplierFunction applier,
   void *context
);
```

Parameters

heap

The binary heap to use.

applier

The callback function to call once for each value in *heap*.

context

A program-defined value that is passed to the *applier* callback function, but is otherwise unused by this function.

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

Declared In
CFBinaryHeap.h

CFBinaryHeapContainsValue

Returns whether a given value is in a binary heap.

```
Boolean CFBinaryHeapContainsValue (
   CFBinaryHeapRef heap,
   const void *value
);
```

Parameters

heap

The binary heap to search.

value

The value for which to find matches in the binary heap. The compare callback provided in the `CFBinaryHeapCallBacks` (page 86) structure when the binary heap was created is used to compare values. If *value*, or any of the values in the binary heap, are not understood by the compare callback, the behavior is undefined.

Return Value
`true` if *value* is a member of *heap*, `false` otherwise.

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

Declared In
CFBinaryHeap.h

CFBinaryHeapCreate

Creates a new mutable or fixed-mutable binary heap.

```
CFBinaryHeapRef CFBinaryHeapCreate (
   CFAllocatorRef allocator,
   CFIndex capacity,
   const CFBinaryHeapCallBacks *callBacks,
   const CFBinaryHeapCompareContext *compareContext
);
```

Parameters

allocator

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

capacity

The maximum number of values that can be contained by the binary heap. The binary heap starts empty and can grow to this number of values. If this parameter is `0`, the binary heap's maximum capacity is limited only by memory.

callBacks

A pointer to a `CFBinaryHeapCallBacks` (page 86) structure initialized with the callbacks that operate on the values placed into the binary heap. If the binary heap will be holding `CFString` objects, pass the `kCFStringBinaryHeapCallBacks` (page 87) constant. This functions makes a copy of the contents of the callbacks structure, so that a pointer to a structure on the stack can be passed in, or can be reused for multiple binary heap creations. This callbacks parameter may not be `NULL`.

compareContext

Not used. Pass `NULL`.

Return Value

A new `CFBinaryHeap` object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBinaryHeap.h`

CFBinaryHeapCreateCopy

Creates a new mutable or fixed-mutable binary heap with the values from a pre-existing binary heap.

```
CFBinaryHeapRef CFBinaryHeapCreateCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFBinaryHeapRef heap
);
```

Parameters

allocator

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

capacity

The maximum number of values that can be contained by the binary heap. The binary heap starts with the same number of values as *heap* and can grow to this number of values. If this parameter is 0, the binary heap's maximum capacity is limited only by memory. If nonzero, *capacity* must be large enough to hold all the values in *heap*.

heap

The binary heap which is to be copied. The values from the binary heap are copied as pointers into the new binary heap (that is, the values themselves are copied, not that to which the values point, if anything). However, the values are also retained by the new binary heap.

Return Value
A new `CFBinaryHeap` object holding the same values as *heap*. The new binary heap uses the same callbacks as *heap*. Ownership follows the Create Rule.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetCount

Returns the number of values currently in a binary heap.

```
CFIndex CFBinaryHeapGetCount (
   CFBinaryHeapRef heap
);
```

Parameters

heap

The binary heap to use.

Return Value
The number of values in *heap*.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetCountOfValue

Counts the number of times a given value occurs in a binary heap.

```
CFIndex CFBinaryHeapGetCountOfValue (
   CFBinaryHeapRef heap,
   const void *value
);
```

Parameters

heap

The binary heap to search.

value

The value for which to find matches in the binary heap. The compare callback provided in the `CFBinaryHeapCallBacks` (page 86) structure when the binary heap was created is used to compare. If *value*, or any of the values in the binary heap, are not understood by the compare callback, the behavior is undefined.

Return Value
The number of times *value* occurs in *heap*.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetMinimum

Returns the minimum value in a binary heap.

```
const void * CFBinaryHeapGetMinimum (
   CFBinaryHeapRef heap
);
```

Parameters

heap

The binary heap to use.

Return Value
The minimum value in *heap* as determined by the binary heap's compare callback. If *heap* contains several equal minimum values, any one may be returned. If the value is a Core Foundation object, ownership follows the Get Rule.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetMinimumIfPresent

Returns the minimum value in a binary heap, if present.

```
Boolean CFBinaryHeapGetMinimumIfPresent (
   CFBinaryHeapRef heap,
   const void **value
);
```

Parameters

heap

The binary heap to use.

value

On return, the minimum value in *heap* as determined by the binary heap's compare callback. If *heap* contains several equal minimum values, any one may be returned. If the value is a Core Foundation object, ownership follows the Get Rule.

Return Value
`true` if a minimum value exists in *heap*, `false` otherwise. `false` is returned only if *heap* is empty.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetTypeID

Returns the type identifier of the `CFBinaryHeap` opaque type.

```
CFTypeID CFBinaryHeapGetTypeID (
   void
);
```

Return Value
The type identifier of the `CFBinaryHeap` opaque type.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapGetValues

Copies all the values from a binary heap into a sorted C array.

```
void CFBinaryHeapGetValues (
   CFBinaryHeapRef heap,
   const void **values
);
```

Parameters

heap
The binary heap to use.

values
On return, the memory pointed to by this argument holds a C array of all the values in heap, sorted from minimum to maximum values. You must allocate sufficient memory to hold all the values in *heap* before calling this function. If the values are Core Foundation objects, ownership follows the Get Rule.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapRemoveAllValues

Removes all values from a binary heap, making it empty.

```
void CFBinaryHeapRemoveAllValues (
   CFBinaryHeapRef heap
);
```

Parameters

heap

The binary heap to use.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapRemoveMinimumValue

Removes the minimum value from a binary heap.

```
void CFBinaryHeapRemoveMinimumValue (
   CFBinaryHeapRef heap
);
```

Parameters

heap

The binary heap to use.

Discussion
If *heap* contains several equal minimum values, only one of them is removed. If *heap* is empty, this function does nothing.

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

Declared In
`CFBinaryHeap.h`

Callbacks

CFBinaryHeapApplierFunction

Callback function used to apply a function to all members of a binary heap.

```
typedef void (*CFBinaryHeapApplierFunction) (
    const void *val,
    void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    const void *val,
    void *context
);
```

Parameters

val

The current value from the binary heap.

context

The program-defined context parameter given to the `CFBinaryHeapApplyFunction` (page 77) function.

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapCompareCallBack

Callback function used to compare two members of a binary heap.

```
typedef CFComparisonResult (*CFBinaryHeapCompareCallBack) (
    const void *ptr1,
    const void *ptr2,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFComparisonResult MyCallBack (
    const void *ptr1,
    const void *ptr2,
    void *info
);
```

Parameters

ptr1

First value to compare.

ptr2

Second value to compare.

info

Not used. Should always be `NULL`.

Return Value
`kCFCompareLessThan` (page 802) if *ptr1* is less than *ptr2*, `kCFCompareEqualTo` (page 802) if *ptr1* and *ptr2* are equal, or `kCFCompareGreaterThan` (page 802) if *ptr1* is greater than *ptr2*.

CFBinaryHeapCopyDescriptionCallBack

Callback function used to get a description of a value in a binary heap.

```
typedef CFStringRef (*CFBinaryHeapCopyDescriptionCallBack) (
    const void *ptr
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
    const void *ptr
);
```

Parameters

ptr

The value to be described.

CFBinaryHeapReleaseCallBack

Callback function used to release a value before it is removed from a binary heap.

```
typedef void (*CFBinaryHeapReleaseCallBack) (
    CFAllocatorRef allocator,
    const void *ptr
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFAllocatorRef allocator,
    const void *ptr
);
```

Parameters

allocator

The binary heap's allocator.

ptr

The value to release.

CFBinaryHeapRetainCallBack

Callback function used to retain a value being added to a binary heap.

```
typedef const void *(*CFBinaryHeapRetainCallBack) (
    CFAllocatorRef allocator,
    const void *ptr
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
    CFAllocatorRef allocator,
    const void *ptr
);
```

Parameters

allocator

The binary heap's allocator.

ptr

The value to retain.

Return Value

The value to store in the binary heap, which is usually the *ptr* parameter passed to this callback, but may be a different value if a different value should be stored in the binary heap.

Data Types

CFBinaryHeapCallBacks

Structure containing the callbacks for values for a `CFBinaryHeap` object.

```
struct CFBinaryHeapCallBacks {
    CFIndex version;
    CFBinaryHeapRetainCallBack retain;
    CFBinaryHeapReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFBinaryHeapCompareCallBack compare;
};
typedef struct CFBinaryHeapCallBacks CFBinaryHeapCallBacks;
```

Fields

`version`

The version number of the structure type being passed in as a parameter to the `CFBinaryHeap` creation functions. This structure is version `0`.

`retain`

The callback used to add a retain for the binary heap on values as they are put into the binary heap. This callback returns the value to use as the value in the binary heap, which is usually the value parameter passed to this callback, but may be a different value if a different value should be added to the binary heap. If this field is `NULL`, the binary heap does nothing to retain a value being added.

`release`

The callback used to remove a retain previously added for the binary heap from values as they are removed from the binary heap. If this field is `NULL`, the binary heap does nothing to release a value being removed.

`copyDescription`

The callback used to create a descriptive string representation of each value in the binary heap. This is used by the `CFCopyDescription` (page 652) function. If this field is `NULL`, the binary heap constructs a `CFString` object describing the value based on its pointer value.

`compare`

The callback used to compare values in the binary heap in some operations. This field cannot be `NULL`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBinaryHeap.h`

CFBinaryHeapCompareContext

Not used.

```
struct CFBinaryHeapCompareContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFBinaryHeapCompareContext CFBinaryHeapCompareContext;
```

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

Declared In
`CFBinaryHeap.h`

CFBinaryHeapRef

A reference to a binary heap object.

```
typedef struct __CFBinaryHeap *CFBinaryHeapRef;
```

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

Declared In
`CFBinaryHeap.h`

Constants

Predefined Callback Structures

`CFBinaryHeap` provides some predefined callbacks for your convenience.

```
const CFBinaryHeapCallBacks kCFStringBinaryHeapCallBacks;
```

Constants

`kCFStringBinaryHeapCallBacks`

Predefined `CFBinaryHeapCallBacks` (page 86) structure containing a set of callbacks appropriate for use when the values in a binary heap are all `CFString` objects.

Available in Mac OS X v10.0 and later.

Declared in `CFBinaryHeap.h`.

CFBitVector Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBitVector.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFBitVector and its derived mutable type, CFMutableBitVector, manage ordered collections of bit values, which are either `0` or `1`. CFBitVector creates static bit vectors and CFMutableBitVector creates dynamic bit vectors.

Functions by Task

Creating a Bit Vector

`CFBitVectorCreate` (page 90)
Creates an immutable bit vector from a block of memory.

`CFBitVectorCreateCopy` (page 91)
Creates an immutable bit vector that is a copy of another bit vector.

Getting Information About a Bit Vector

`CFBitVectorContainsBit` (page 90)
Returns whether a bit vector contains a particular bit value.

`CFBitVectorGetBitAtIndex` (page 92)
Returns the bit value at a given index in a bit vector.

`CFBitVectorGetBits` (page 92)
Returns the bit values in a range of indices in a bit vector.

`CFBitVectorGetCount` (page 92)
Returns the number of bit values in a bit vector.

`CFBitVectorGetCountOfBit` (page 93)
Counts the number of times a certain bit value occurs within a range of bits in a bit vector.

`CFBitVectorGetFirstIndexOfBit` (page 93)
Locates the first occurrence of a certain bit value within a range of bits in a bit vector.

`CFBitVectorGetLastIndexOfBit` (page 94)
Locates the last occurrence of a certain bit value within a range of bits in a bit vector.

Getting the CFBitVector Type ID

`CFBitVectorGetTypeID` (page 95)
Returns the type identifier for the CFBitVector opaque type.

Functions

CFBitVectorContainsBit

Returns whether a bit vector contains a particular bit value.

```
Boolean CFBitVectorContainsBit (
   CFBitVectorRef bv,
   CFRange range,
   CFBit value
);
```

Parameters

bv
The bit vector to search.

range
The range of bits in *bv* to search.

value
The bit value for which to search.

Return Value
`true` if the specified range of bits in *bv* contains *value*, otherwise `false`.

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

Declared In
`CFBitVector.h`

CFBitVectorCreate

Creates an immutable bit vector from a block of memory.

```
CFBitVectorRef CFBitVectorCreate (
   CFAllocatorRef allocator,
   const UInt8 *bytes,
   CFIndex numBits
);
```

Parameters

allocator

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

bytes

A pointer to the bit values to store in the new bit vector. The values are copied into the bit vector's own memory. The bit indices are numbered left-to-right with `0` being the left-most, or most-significant, bit in the byte stream.

numBits

The number of bits in the bit vector.

Return Value
A new bit vector. Ownership follows the Create Rule.

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

Declared In
`CFBitVector.h`

CFBitVectorCreateCopy

Creates an immutable bit vector that is a copy of another bit vector.

```
CFBitVectorRef CFBitVectorCreateCopy (
   CFAllocatorRef allocator,
   CFBitVectorRef bv
);
```

Parameters

allocator

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

bv

The bit vector to copy.

Return Value
A new bit vector holding the same bit values as *bv*. Ownership follows the Create Rule.

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

Declared In
`CFBitVector.h`

CFBitVectorGetBitAtIndex

Returns the bit value at a given index in a bit vector.

```
CFBit CFBitVectorGetBitAtIndex (
   CFBitVectorRef bv,
   CFIndex idx
);
```

Parameters

bv

The bit vector to examine.

idx

The index of the bit value in *bv* to return.

Return Value

The bit value at index *idx* in *bv*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFBitVector.h

CFBitVectorGetBits

Returns the bit values in a range of indices in a bit vector.

```
void CFBitVectorGetBits (
   CFBitVectorRef bv,
   CFRange range,
   UInt8 *bytes
);
```

Parameters

bv

The bit vector to examine.

range

The range of bit values to return.

bytes

On return, contains the requested bit values from *bv*. This argument must point to enough memory to hold the number of bits requested. The requested bits are left-aligned with the first requested bit stored in the left-most, or most-significant, bit of the byte stream.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFBitVector.h

CFBitVectorGetCount

Returns the number of bit values in a bit vector.

```
CFIndex CFBitVectorGetCount (
   CFBitVectorRef bv
);
```

Parameters

bv

The bit vector to examine.

Return Value
The current size of *bv*.

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

Declared In
CFBitVector.h

CFBitVectorGetCountOfBit

Counts the number of times a certain bit value occurs within a range of bits in a bit vector.

```
CFIndex CFBitVectorGetCountOfBit (
   CFBitVectorRef bv,
   CFRange range,
   CFBit value
);
```

Parameters

bv

The bit vector to examine.

range

The range of bits in *bv* to search.

value

The bit value to count.

Return Value
The number of occurrences of *value* in the specified range of *bv*.

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

Declared In
CFBitVector.h

CFBitVectorGetFirstIndexOfBit

Locates the first occurrence of a certain bit value within a range of bits in a bit vector.

```
CFIndex CFBitVectorGetFirstIndexOfBit (
   CFBitVectorRef bv,
   CFRange range,
   CFBit value
);
```

Parameters

bv

The bit vector to examine.

range

The range of bits in *bv* to search.

value

The bit value for which to search.

Return Value

The index of the first occurrence of *value* in the specified range of *bv*, or `kCFNotFound` if *value* is not present.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBitVector.h`

CFBitVectorGetLastIndexOfBit

Locates the last occurrence of a certain bit value within a range of bits in a bit vector.

```
CFIndex CFBitVectorGetLastIndexOfBit (
   CFBitVectorRef bv,
   CFRange range,
   CFBit value
);
```

Parameters

bv

The bit vector to examine.

range

The range of bits in *bv* to search.

value

The bit value for which to search.

Return Value

The index of the last occurrence of *value* in the specified range of *bv*, or `kCFNotFound` if *value* is not present.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBitVector.h`

CFBitVectorGetTypeID

Returns the type identifier for the CFBitVector opaque type.

```
CFTypeID CFBitVectorGetTypeID (
   void
);
```

Return Value
The type identifier for the CFBitVector opaque type.

Discussion
CFMutableBitVector objects have the same type identifier as CFBitVector objects.

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

Declared In
`CFBitVector.h`

Data Types

CFBit

A binary value of either `0` or `1`.

```
typedef UInt32 CFBit;
```

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

Declared In
`CFBitVector.h`

CFBitVectorRef

A reference to an immutable bit vector object.

```
typedef const struct __CFBitVector *CFBitVectorRef;
```

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

Declared In
`CFBitVector.h`

CFBoolean Reference

Derived From:	CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFNumber.h
Companion guide	Property List Programming Topics for Core Foundation

Overview

CFBoolean objects are used to wrap boolean values for use in Core Foundation property lists and collection types.

Functions

CFBooleanGetTypeID

Returns the Core Foundation type identifier for the CFBoolean opaque type.

```
CFTypeID CFBooleanGetTypeID (
   void
);
```

Return Value
The Core Foundation type identifier for CFBoolean opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
Cocoa PDE with Carbon Printing
HID Manager Basics
LoginItemsAE

Declared In
`CFNumber.h`

CFBooleanGetValue

Returns the value of a CFBoolean object as a standard C type `Boolean`.

```
Boolean CFBooleanGetValue (
   CFBooleanRef boolean
);
```

Parameters

boolean

The boolean to examine.

Return Value

The value of *boolean*.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

audioburntest

databurntest

GLUT

HID Utilities Source

SeeMyFriends

Declared In

`CFNumber.h`

Data Types

CFBooleanRef

A reference to a CFBoolean object.

```
typedef const struct __CFBoolean *CFBooleanRef;
```

Discussion

CFBoolean objects are used to wrap boolean values for use in Core Foundation property lists and collection types.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFNumber.h`

Constants

Boolean Values

CFBoolean evaluates to either true or false values where `kCFBooleanTrue` is the true, and `kCFBooleanFalse` is the false value.

```
const CFBooleanRef kCFBooleanTrue;
const CFBooleanRef kCFBooleanFalse;
```

Constants

`kCFBooleanTrue`

Boolean true value.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFBooleanFalse`

Boolean false value.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

CFBundle Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBundle.h
Companion guides	Bundle Programming Guide Plug-ins

Overview

CFBundle allows you to use a folder hierarchy called a bundle to organize and locate many types of application resources including images, sounds, localized strings, and executable code. In Mac OS X, bundles can also be used by CFM applications to load and execute functions from Mach-O frameworks. You can use bundles to support multiple languages or execute your application on multiple operating environments.

You create a bundle object using one of the `CFBundleCreate...` functions. CFBundle provides several functions for finding resources within a bundle. The `CFBundleCopyResourceURL` (page 114) function returns the location of a resource of the specified name and type, and in the specified subdirectory. Use `CFBundleCopyResourceURLForLocalization` (page 115) to restrict the search to a specific localization name. Use `CFBundleCopyResourceURLsOfType` (page 117) to get the locations of all resources of a specified type.

CFBundle provides functions for getting bundle information, such as its identifier and information dictionary. Use the `CFBundleGetIdentifier` (page 126) function to get the identifier of a bundle, and the `CFBundleGetInfoDictionary` (page 127) function to get its information dictionary. The principal intended purpose for locating bundles by identifier is so that code (in frameworks, plugins, etc.) can find its own bundle.

You can also obtain locations of subdirectories in a bundle represented as CFURL objects. The `CFBundleCopyExecutableURL` (page 109) function returns the location of the application's executable. The functions `CFBundleCopyResourceURL` (page 114), `CFBundleCopySharedFrameworksURL` (page 119), `CFBundleCopyPrivateFrameworksURL` (page 113), `CFBundleCopySharedSupportURL` (page 120), and `CFBundleCopyBuiltInPlugInsURL` (page 107) return the location of a bundle's subdirectory containing resources, shared frameworks, private frameworks, shared support files, and plug-ins respectively.

Other functions are used to manage localizations. The `CFBundleCopyLocalizedString` (page 112) and `CFBundleCopyLocalizationsForURL` (page 111) functions return a localized string from a bundle's strings file. The `CFBundleCopyLocalizationsForPreferences` (page 111) function returns the localizations that CFBundle would prefer, given the specified bundle and user preference localizations.

Unlike some other Core Foundation opaque types with similar Cocoa Foundation names (such as CFString and `NSString`), `NSBundle` objects cannot be cast ("toll-free bridged") to CFBundle objects.

Unlike NSBundle, which does not support unloading (because the Objective C runtime does not support the unloading of Objective C code), you can unload CFBundle objects.

`CFBundleGetFunctionPointerForName` (page 125) and related calls automatically load a bundle if it is not already loaded. When the last reference to the CFBundle object is released and it is finally deallocated, then the code will be unloaded if it is still loaded and if the executable is of a type that supports unloading. If you keep this in mind, and if you make sure that everything that uses the bundle keeps a retain on the CFBundle object, then you can just use the bundle naturally and never have to worry about when it is loaded and unloaded.

On the other hand, if you want to manually manage when the bundle is loaded and unloaded, then you can use `CFBundleLoadExecutable` (page 132) and `CFBundleUnloadExecutable` (page 134)—although this technique is not recommended. These functions force immediate loading and unloading of the executable (if it has not already been loaded/unloaded, and in the case of unloading if the executable is of a type that supports unloading). If you do this, then the code calling `CFBundleUnloadExecutable` is responsible for making sure that there are no remaining references to anything in the bundle's code before it is unloaded. In the previous approach, by contrast, this responsibility can be distributed to the individual code sections that use the bundle, by making sure that each one keeps its own retain on the CFBundle object.

One further point about CFBundle reference counting: if you are taking the first approach, but do not actually wish the bundle's code to be unloaded (as is often the case), or if you are taking the second approach of manually managing the unloading yourself, then in many cases you do not actually have to worry about releasing a CFBundle object. CFBundle instances are uniqued, so there is only one CFBundle object for a given bundle, and rarely are there so many bundles being considered at once that the memory usage for CFBundle objects would be significant. There are cases in which a process could create CFBundle objects for potentially an unlimited number of bundles, and such processes would wish to balance retains and releases carefully, but such cases are likely to be rare.

Note that it is best to compile any unloadable bundles with the flag `-fno-constant-cfstrings`—see *Bundle Programming Guide* for more details.

Functions by Task

Creating and Accessing Bundles

`CFBundleCreate` (page 121)
: Creates a CFBundle object.

`CFBundleCreateBundlesFromDirectory` (page 122)
: Searches a directory and constructs an array of CFBundle objects from all valid bundles in the specified directory.

`CFBundleGetAllBundles` (page 122)
: Returns an array containing all of the bundles currently open in the application.

`CFBundleGetBundleWithIdentifier` (page 123)
: Locate a bundle given its program-defined identifier.

`CFBundleGetMainBundle` (page 128)
: Returns an application's main bundle.

Loading and Unloading a Bundle

`CFBundleIsExecutableLoaded` (page 131)
Obtains information about the load status for a bundle's main executable.

`CFBundlePreflightExecutable` (page 134)
Returns a Boolean value that indicates whether a given bundle is loaded or appears to be loadable.

`CFBundleLoadExecutable` (page 132)
Loads a bundle's main executable code into memory and dynamically links it into the running application.

`CFBundleLoadExecutableAndReturnError` (page 132)
Returns a Boolean value that indicates whether a given bundle is loaded, attempting to load it if necessary.

`CFBundleUnloadExecutable` (page 134)
Unloads the main executable for the specified bundle.

Finding Locations in a Bundle

`CFBundleCopyAuxiliaryExecutableURL` (page 106)
Returns the location of a bundle's auxiliary executable code.

`CFBundleCopyBuiltInPlugInsURL` (page 107)
Returns the location of a bundle's built in plug-in.

`CFBundleCopyExecutableURL` (page 109)
Returns the location of a bundle's main executable code.

`CFBundleCopyPrivateFrameworksURL` (page 113)
Returns the location of a bundle's private Frameworks directory.

`CFBundleCopyResourcesDirectoryURL` (page 114)
Returns the location of a bundle's Resources directory.

`CFBundleCopySharedFrameworksURL` (page 119)
Returns the location of a bundle's shared frameworks directory.

`CFBundleCopySharedSupportURL` (page 120)
Returns the location of a bundle's shared support files directory.

`CFBundleCopySupportFilesDirectoryURL` (page 120)
Returns the location of the bundle's support files directory.

Locating Bundle Resources

`CFBundleCloseBundleResourceMap` (page 106)
Closes an open resource map for a bundle.

`CFBundleCopyResourceURL` (page 114)
Returns the location of a resource contained in the specified bundle.

`CFBundleCopyResourceURLInDirectory` (page 116)
Returns the location of a resource contained in the specified bundle directory without requiring the creation of a CFBundle object.

`CFBundleCopyResourceURLsOfType` (page 117)
Assembles an array of URLs specifying all of the resources of the specified type found in a bundle.

`CFBundleCopyResourceURLsOfTypeInDirectory` (page 119)
Returns an array of CFURL objects describing the locations of all resources in a bundle of the specified type without needing to create a CFBundle object.

`CFBundleCopyResourceURLForLocalization` (page 115)
Returns the location of a localized resource in a bundle.

`CFBundleCopyResourceURLsOfTypeForLocalization` (page 118)
Returns an array containing copies of the URL locations for a specified bundle, resource, and localization name.

`CFBundleOpenBundleResourceFiles` (page 133)
Opens the non-localized and localized resource files (if any) for a bundle in separate resource maps.

`CFBundleOpenBundleResourceMap` (page 133)
Opens the non-localized and localized resource files (if any) for a bundle in a single resource map.

Managing Localizations

`CFBundleCopyBundleLocalizations` (page 107)
Returns an array containing a bundle's localizations.

`CFBundleCopyLocalizedString` (page 112)
Returns a localized string from a bundle's strings file.

`CFBundleCopyLocalizationsForPreferences` (page 111)
Given an array of possible localizations and preferred locations, returns the one or more of them that CFBundle would use, without reference to the current application context.

`CFBundleCopyLocalizationsForURL` (page 111)
Returns an array containing the localizations for a bundle or executable at a particular location.

`CFBundleCopyPreferredLocalizationsFromArray` (page 113)
Given an array of possible localizations, returns the one or more of them that CFBundle would use in the current application context.

`CFCopyLocalizedString` (page 135)
Searches the default strings file `Localizable.strings` for the string associated with the specified key.

`CFCopyLocalizedStringFromTable` (page 135)
Searches the specified strings file for the string associated with the specified key.

`CFCopyLocalizedStringFromTableInBundle` (page 136)
Returns a localized version of the specified string.

`CFCopyLocalizedStringWithDefaultValue` (page 137)
Returns a localized version of a localization string.

Managing Executable Code

`CFBundleGetDataPointerForName` (page 123)
Returns a data pointer to a symbol of the given name.

`CFBundleGetDataPointersForNames` (page 124)
Returns a C array of data pointer to symbols of the given names.

`CFBundleGetFunctionPointerForName` (page 125)
Returns a pointer to a function in a bundle's executable code using the function name as the search key.

`CFBundleGetFunctionPointersForNames` (page 126)
Constructs a function table containing pointers to all of the functions found in a bundle's main executable code.

`CFBundleGetPlugIn` (page 129)
Returns a bundle's plug-in.

Getting Bundle Properties

`CFBundleCopyBundleURL` (page 108)
Returns the location of a bundle.

`CFBundleGetDevelopmentRegion` (page 124)
Returns the bundle's development region from the bundle's information property list.

`CFBundleGetIdentifier` (page 126)
Returns the bundle identifier from a bundle's information property list.

`CFBundleGetInfoDictionary` (page 127)
Returns a bundle's information dictionary.

`CFBundleGetLocalInfoDictionary` (page 127)
Returns a bundle's localized information dictionary.

`CFBundleGetValueForInfoDictionaryKey` (page 130)
Returns a value (localized if possible) from a bundle's information dictionary.

`CFBundleCopyInfoDictionaryInDirectory` (page 110)
Returns a bundle's information dictionary.

`CFBundleCopyInfoDictionaryForURL` (page 110)
Returns the information dictionary for a given URL location.

`CFBundleGetPackageInfo` (page 128)
Returns a bundle's package type and creator.

`CFBundleGetPackageInfoInDirectory` (page 129)
Returns a bundle's package type and creator without having to create a CFBundle object.

`CFBundleCopyExecutableArchitectures` (page 108)
Returns an array of CFNumbers representing the architectures a given bundle provides.

`CFBundleCopyExecutableArchitecturesForURL` (page 109)
Returns an array of CFNumbers representing the architectures a given URL provides.

`CFBundleGetVersionNumber` (page 130)
Returns a bundle's version number.

Getting the CFBundle Type ID

`CFBundleGetTypeID` (page 130)
Returns the type identifier for the CFBundle opaque type.

Functions

CFBundleCloseBundleResourceMap

Closes an open resource map for a bundle.

```
void CFBundleCloseBundleResourceMap (
   CFBundleRef bundle,
   CFBundleRefNum refNum
);
```

Parameters

bundle

The bundle whose resource map is referenced by *refNum*.

refNum

The reference number for a resource map to close.

Discussion

You open a resource map using either `CFBundleOpenBundleResourceFiles` (page 133) or `CFBundleOpenBundleResourceMap` (page 133).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleCopyAuxiliaryExecutableURL

Returns the location of a bundle's auxiliary executable code.

```
CFURLRef CFBundleCopyAuxiliaryExecutableURL (
   CFBundleRef bundle,
   CFStringRef executableName
);
```

Parameters

bundle

The bundle to examine.

executableName

The name of *bundle*'s auxiliary executable code.

Return Value

The URL location of the specified bundle's auxiliary executable code, or `NULL` if it could not be found. Ownership follows the Create Rule.

Discussion

This function can be used to find executables other than your main executable. This is useful, for instance, for applications that have some command line tool that is packaged with and used by the application. The tool can be packaged in the various platform executable directories in the bundle and can be located with this function. This allows an application to ship versions of the tool for each platform as it does for the main application executable.

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

Related Sample Code
BackgroundExporter
BetterAuthorizationSample
BSDLLCTest

Declared In
`CFBundle.h`

CFBundleCopyBuiltInPlugInsURL

Returns the location of a bundle's built in plug-in.

```
CFURLRef CFBundleCopyBuiltInPlugInsURL (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value
A CFURL object describing the location of *bundle*'s built in plug-ins, or `NULL` if it could not be found. Ownership follows the Create Rule.

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

Related Sample Code
BasicPlugIn

Declared In
`CFBundle.h`

CFBundleCopyBundleLocalizations

Returns an array containing a bundle's localizations.

```
CFArrayRef CFBundleCopyBundleLocalizations (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value
An array containing *bundle*'s localizations. Ownership follows the Create Rule.

Discussion
The array returned by this function is typically passed as a parameter to either the `CFBundleCopyPreferredLocalizationsFromArray` (page 113) or `CFBundleCopyLocalizationsForPreferences` (page 111) function.

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

Declared In
`CFBundle.h`

CFBundleCopyBundleURL

Returns the location of a bundle.

```
CFURLRef CFBundleCopyBundleURL (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle to examine.

Return Value
A CFURL object describing the location of *bundle*, or `NULL` if the specified bundle does not exist. Ownership follows the Create Rule.

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

Related Sample Code
HID Utilities Source
simpleJavaLauncher

Declared In
`CFBundle.h`

CFBundleCopyExecutableArchitectures

Returns an array of CFNumbers representing the architectures a given bundle provides.

```
CFArrayRef CFBundleCopyExecutableArchitectures (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle to examine.

Return Value
If the bundle's executable exists and is a Mach-O file, returns an array of CFNumbers whose values are integers representing the architectures the file provides. Possible values are listed in "Architecture Types" (page 140). If the executable is not a Mach-O file, returns `NULL`. Ownership follows the Create Rule.

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

See Also
`CFBundleCopyExecutableArchitecturesForURL` (page 109)

Declared In
`CFBundle.h`

CFBundleCopyExecutableArchitecturesForURL

Returns an array of CFNumbers representing the architectures a given URL provides.

```
CFArrayRef CFBundleCopyExecutableArchitecturesForURL (
   CFURLRef url
);
```

Parameters

url

The URL to examine.

Return Value
For a directory URL, if the bundle's executable exists and is a Mach-O file, returns an array of CFNumbers whose values are integers representing the architectures the URL provides. For a plain file URL representing an unbundled executable, returns the architectures it provides if it is a Mach-O file. Possible values are listed in "Architecture Types" (page 140). If there is no bundle executable or if the executable is not a Mach-O file, returns `NULL`. Ownership follows the Create Rule.

Discussion
For a directory URL, this is equivalent to calling `CFBundleCopyExecutableArchitectures` (page 108) on the corresponding bundle.

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

See Also
`CFBundleCopyExecutableArchitectures` (page 108)

Declared In
`CFBundle.h`

CFBundleCopyExecutableURL

Returns the location of a bundle's main executable code.

```
CFURLRef CFBundleCopyExecutableURL (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value
A CFURL object describing the location of *bundle*'s executable code, or `NULL` if none is found. Ownership follows the Create Rule.

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

Related Sample Code
CheckExecutableArchitecture
MemoryBasedBundle

Declared In
`CFBundle.h`

CFBundleCopyInfoDictionaryForURL

Returns the information dictionary for a given URL location.

```
CFDictionaryRef CFBundleCopyInfoDictionaryForURL (
   CFURLRef url
);
```

Parameters

url
A CFURL object describing the location of a file.

Return Value
A CFDictionary object containing *url*'s information dictionary. Ownership follows the Create Rule.

Discussion
For a directory URL, this is equivalent to `CFBundleCopyInfoDictionaryInDirectory` (page 110). For a plain file URL representing an unbundled application, this function will attempt to read an information dictionary either from the `(__TEXT, __info_plist)` section of the file (for a Mach-O file) or from a `plst` resource.

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

Declared In
`CFBundle.h`

CFBundleCopyInfoDictionaryInDirectory

Returns a bundle's information dictionary.

```
CFDictionaryRef CFBundleCopyInfoDictionaryInDirectory (
   CFURLRef bundleURL
);
```

Parameters

bundleURL
A CFURL object describing the location of a bundle.

Return Value
A CFDictionary object containing the information dictionary for a bundle located at *bundleURL*. Ownership follows the Create Rule.

Discussion
This function provides a means to obtain an information dictionary for a bundle without first creating a CFBundle object.

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

Declared In
CFBundle.h

CFBundleCopyLocalizationsForPreferences

Given an array of possible localizations and preferred locations, returns the one or more of them that CFBundle would use, without reference to the current application context.

```
CFArrayRef CFBundleCopyLocalizationsForPreferences (
   CFArrayRef locArray,
   CFArrayRef prefArray
);
```

Parameters

locArray
An array of possible localizations to search.

prefArray
An array of preferred localizations. If NULL, the user's actual preferred localizations will be used.

Return Value
An array containing the localizations that CFBundle would use. Ownership follows the Create Rule.

Discussion
This is not the same as CFBundleCopyPreferredLocalizationsFromArray (page 113), because that function takes the current application context into account. To determine the localizations that another application would use, apply this function to the result of CFBundleCopyBundleLocalizations (page 107).

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

Declared In
CFBundle.h

CFBundleCopyLocalizationsForURL

Returns an array containing the localizations for a bundle or executable at a particular location.

```
CFArrayRef CFBundleCopyLocalizationsForURL (
   CFURLRef url
);
```

Parameters

url

The location of a bundle's localizations.

Return Value

An array containing the localizations available at *url*. Ownership follows the Create Rule.

Discussion

For a directory URL, this is equivalent to calling the `CFBundleCopyBundleLocalizations` (page 107) function on the corresponding bundle. For a plain file URL representing an unbundled application, this will attempt to determine its localizations using the `kCFBundleLocalizationsKey` (page 139) and `kCFBundleDevelopmentRegionKey` (page 139) keys in the dictionary returned by `CFBundleCopyInfoDictionaryForURL` (page 110), or a `vers` resource if those are not present.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFBundle.h`

CFBundleCopyLocalizedString

Returns a localized string from a bundle's strings file.

```
CFStringRef CFBundleCopyLocalizedString (
   CFBundleRef bundle,
   CFStringRef key,
   CFStringRef value,
   CFStringRef tableName
);
```

Parameters

bundle

The bundle to examine.

key

The key for the localized string to retrieve. This key will be used to look up the localized string in the strings file. Typically the key is identical to the value of the localized string in the development language.

value

A default value to return if no value exists for *key*.

tableName

The name of the strings file to search. The name should not include the `strings` filename extension. The case of the string must match that of the file name, even on file systems (such as HFS+) that are not case sensitive with regards to file names

Return Value

A CFString object that contains the localized string. If no value exists for *key*, returns *value* unless *value* is `NULL` or an empty string, in which case *key* is returned instead. Ownership follows the Create Rule.

Discussion
This is the base function from which the other localized string macros are derived. In general you should not use this function because the `genstrings` development tool only recognizes the macro version of this call when generating strings files. See `CFCopyLocalizedString` (page 135) for details on how to use these macros.

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

Related Sample Code
MoreSCF

Declared In
`CFBundle.h`

CFBundleCopyPreferredLocalizationsFromArray

Given an array of possible localizations, returns the one or more of them that CFBundle would use in the current application context.

```
CFArrayRef CFBundleCopyPreferredLocalizationsFromArray (
   CFArrayRef locArray
);
```

Parameters

`locArray`
An array of possible localizations.

Return Value
A subset of *`locArray`* that CFBundle would use in the current application context. Ownership follows the Create Rule.

Discussion
You can obtain *`locArray`* using the `CFBundleCopyBundleLocalizations` (page 107) function.

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

Declared In
`CFBundle.h`

CFBundleCopyPrivateFrameworksURL

Returns the location of a bundle's private Frameworks directory.

```
CFURLRef CFBundleCopyPrivateFrameworksURL (
   CFBundleRef bundle
);
```

Parameters

`bundle`
The bundle to examine.

Return Value
A CFURL object describing the location of *bundle*'s private frameworks directory, or `NULL` if it could not be found. Ownership follows the Create Rule.

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

Related Sample Code
SpellingChecker CarbonCocoa Bundled
SpellingChecker-CarbonCocoa

Declared In
`CFBundle.h`

CFBundleCopyResourcesDirectoryURL

Returns the location of a bundle's Resources directory.

```
CFURLRef CFBundleCopyResourcesDirectoryURL (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle to examine.

Return Value
A CFURL object describing the location of *bundle*'s resources directory, or `NULL` if it could not be found. Ownership follows the Create Rule.

Discussion
In general, you should never need to use this function. Use `CFBundleCopyResourceURL` (page 114) and similar functions instead.

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

Related Sample Code
FileNotification
PBORenderToVertexArray

Declared In
`CFBundle.h`

CFBundleCopyResourceURL

Returns the location of a resource contained in the specified bundle.

```
CFURLRef CFBundleCopyResourceURL (
   CFBundleRef bundle,
   CFStringRef resourceName,
   CFStringRef resourceType,
   CFStringRef subDirName
);
```

Parameters

bundle

The bundle to examine.

resourceName

The name of the requested resource.

resourceType

The abstract type of the requested resource. The type is expressed as a filename extension, such as `jpg`. Pass `NULL` if you don't need to search by type.

subDirName

The name of the subdirectory of the bundle's resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

Return Value

A CFURL object describing the location of the requested resource, or `NULL` if the resource cannot be found. Ownership follows the Create Rule.

Discussion

For example, if a bundle contains a subdirectory `WaterSounds` that includes a file `Water1.aiff`, you can retrieve the URL for the file using:

```
CFBundleCopyResourceURL(bundle, CFSTR("Water1"), CFSTR("aiff"),
CFSTR("WaterSounds"));
```

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CoreTextTest

HID Config Save

HID Dumper

HID Utilities

SampleDS

Declared In

`CFBundle.h`

CFBundleCopyResourceURLForLocalization

Returns the location of a localized resource in a bundle.

```
CFURLRef CFBundleCopyResourceURLForLocalization (
   CFBundleRef bundle,
   CFStringRef resourceName,
   CFStringRef resourceType,
   CFStringRef subDirName,
   CFStringRef localizationName
);
```

Parameters

bundle

The bundle to examine.

resourceName

The name of the requested resource.

resourceType

The abstract type of the resource to locate. The type is expressed as a filename extension, such as `jpg`.

subDirName

The name of the subdirectory of the bundle's resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

localizationName

The name of the localization. This value should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension. (This parameter is treated literally: If you pass `"de"`, the function will not match resources in a `German.lproj` directory in the bundle.)

Return Value

The location of a localized resource in *bundle*, or `NULL` if the resource could not be found. Ownership follows the Create Rule.

Discussion

Note that file names are case-sensitive, even on file systems (such as HFS+) that are not case sensitive with regards to file names.

You should typically have little reason to use this function (see Getting the Current Language and Locale)—CFBundle's interfaces automatically apply the user's preferences to determine which localized resource files to return in response to a programmatic request. See also `CFBundleCopyBundleLocalizations` (page 107) for how to determine what localizations are available

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleCopyResourceURLInDirectory

Returns the location of a resource contained in the specified bundle directory without requiring the creation of a CFBundle object.

```
CFURLRef CFBundleCopyResourceURLInDirectory (
   CFURLRef bundleURL,
   CFStringRef resourceName,
   CFStringRef resourceType,
   CFStringRef subDirName
);
```

Parameters

bundleURL

The bundle to examine.

resourceName

The name of the requested resource.

resourceType

The abstract type of the requested resource. The type is expressed as a filename extension, such as `jpg`. Pass `NULL` if you don't need to search by type.

subDirName

The name of the subdirectory of the bundle's resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

Return Value

A CFURL object describing the location of the requested resource, or `NULL` if the resource cannot be found. Ownership follows the Create Rule.

Discussion

This function provides a means to obtain package information for a bundle without first creating a bundle. However, since CFBundle objects cache search results, it is faster to create a CFBundle object if you need to repeatedly access resources.

Note that searches are case-sensitive, even on file systems (such as HFS+) that are not case sensitive with regards to file names.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleCopyResourceURLsOfType

Assembles an array of URLs specifying all of the resources of the specified type found in a bundle.

```
CFArrayRef CFBundleCopyResourceURLsOfType (
   CFBundleRef bundle,
   CFStringRef resourceType,
   CFStringRef subDirName
);
```

Parameters

bundle

The bundle to examine.

resourceType

The abstract type of the resources to locate. The type is expressed as a filename extension, such as `jpg`.

subDirName

The name of the subdirectory of the bundle's resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

Return Value
A CFArray object containing CFURL objects of the requested resources. Ownership follows the Create Rule.

Discussion
Note that searches are case-sensitive, even on file systems (such as HFS+) that are not case sensitive with regards to file names.

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

Declared In
`CFBundle.h`

CFBundleCopyResourceURLsOfTypeForLocalization

Returns an array containing copies of the URL locations for a specified bundle, resource, and localization name.

```
CFArrayRef CFBundleCopyResourceURLsOfTypeForLocalization (
   CFBundleRef bundle,
   CFStringRef resourceType,
   CFStringRef subDirName,
   CFStringRef localizationName
);
```

Parameters

bundle

The bundle to examine.

resourceType

The abstract type of the resources to locate. The type is expressed as a filename extension, such as `jpg`.

subDirName

The name of the subdirectory of the bundle's Resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

localizationName

The name of the localization. This value should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension. (This parameter is treated literally: If you pass `"de"`, the function will not match resources in a `German.lproj` directory in the bundle.)

Return Value
A CFArray object containing copies of the requested locations. Ownership follows the Create Rule.

Discussion
Note that file names are case-sensitive, even on file systems (such as HFS+) that are not case sensitive with regards to file names.

You should typically have little reason to use this function (see Getting the Current Language and Locale)—CFBundle's interfaces automatically apply the user's preferences to determine which localized resource files to return in response to a programmatic request. See also `CFBundleCopyBundleLocalizations` (page 107) for how to determine what localizations are available

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

Declared In
`CFBundle.h`

CFBundleCopyResourceURLsOfTypeInDirectory

Returns an array of CFURL objects describing the locations of all resources in a bundle of the specified type without needing to create a CFBundle object.

```
CFArrayRef CFBundleCopyResourceURLsOfTypeInDirectory (
   CFURLRef bundleURL,
   CFStringRef resourceType,
   CFStringRef subDirName
);
```

Parameters

`bundleURL`
The location of a bundle to examine.

`resourceType`
The abstract type of the resources to locate. The type is expressed as a filename extension, such as `jpg`.

`subDirName`
The name of the subdirectory of the bundle's resources directory to search. Pass `NULL` to search the standard CFBundle resource locations.

Return Value
A CFArray object containing the CFURL objects of the requested resources. Ownership follows the Create Rule.

Discussion
This function provides a means to obtain an array containing the locations of all of the requested resources without first creating a CFBundle object. However, since CFBundle objects cache search results, it is faster to create a CFBundle object if you need to repeatedly access resources.

Note that file names are case-sensitive, even on file systems (such as HFS+) that are not case sensitive with regards to file names.

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

Declared In
`CFBundle.h`

CFBundleCopySharedFrameworksURL

Returns the location of a bundle's shared frameworks directory.

```
CFURLRef CFBundleCopySharedFrameworksURL (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

A CFURL object containing the location of *bundle*'s shared frameworks directory, or `NULL` if it could not be found. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleCopySharedSupportURL

Returns the location of a bundle's shared support files directory.

```
CFURLRef CFBundleCopySharedSupportURL (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

A CFURL object containing the location of *bundle*'s shared support files directory, or `NULL` if it could not be found. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BasicInputMethod

Declared In

`CFBundle.h`

CFBundleCopySupportFilesDirectoryURL

Returns the location of the bundle's support files directory.

```
CFURLRef CFBundleCopySupportFilesDirectoryURL (
   CFBundleRef bundle
);
```

Parameters

bundle

The CFBundle object whose support files directory you want to locate.

Return Value
A CFURL object describing the location of the bundle's support files directory, or `NULL` if it could not be found. Ownership follows the Create Rule.

Discussion
In general, you should never need to use this function. Use `CFBundleCopyResourceURL` (page 114) and similar functions instead.

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

Declared In
`CFBundle.h`

CFBundleCreate

Creates a CFBundle object.

```
CFBundleRef CFBundleCreate (
   CFAllocatorRef allocator,
   CFURLRef bundleURL
);
```

Parameters

`allocator`
The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

`bundleURL`
The location of the bundle for which to create a CFBundle object.

Return Value
A CFBundle object created from the bundle at *`bundleURL`*. Ownership follows the Create Rule.

Returns `NULL` if there was a memory allocation problem. May return an existing CFBundle object with the reference count incremented. May return `NULL` if the bundle doesn't exist at *`bundleURL`* (see Discussion).

Discussion
Once a bundle has been created, it is cached; the bundle cache is flushed only periodically. `CFBundleCreate` does not check that a cached bundle still exists in the filesystem. If a bundle is deleted from the filesystem, it is therefore possible for `CFBundleCreate` to return a cached bundle that has actually been deleted.

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

Related Sample Code
CheckExecutableArchitecture
HID Utilities Source
MemoryBasedBundle
SpellingChecker CarbonCocoa Bundled
SpellingChecker-CarbonCocoa

Declared In
`CFBundle.h`

CFBundleCreateBundlesFromDirectory

Searches a directory and constructs an array of CFBundle objects from all valid bundles in the specified directory.

```
CFArrayRef CFBundleCreateBundlesFromDirectory (
   CFAllocatorRef allocator,
   CFURLRef directoryURL,
   CFStringRef bundleType
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

directoryURL

The location of the directory to search for valid bundles.

bundleType

The abstract type of the bundles to locate and create. The type is expressed as a filename extension, such as `bundle`. Pass `NULL` to create CFBundle objects for bundles of any type.

Return Value

A CFArray object containing CFBundle objects created from the contents of the specified directory. Returns an empty array if no bundles exist at *directoryURL*, and `NULL` if there was a memory allocation problem. Ownership follows the Create Rule.

Discussion

The array returned by this function will not contain stale CFBundle references.

Special Considerations

The Create Rule applies both to the array returned and to the bundles in the array. In order to properly dispose of the returned value, you must release the array *and* any bundles returned in the array.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleGetAllBundles

Returns an array containing all of the bundles currently open in the application.

```
CFArrayRef CFBundleGetAllBundles (
   void
);
```

Return Value

A CFArray object containing CFBundle objects for each open bundle in the application. Ownership follows the Get Rule.

Discussion

This function is potentially expensive, so use with care.

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

Related Sample Code
HID Utilities

Declared In
`CFBundle.h`

CFBundleGetBundleWithIdentifier

Locate a bundle given its program-defined identifier.

```
CFBundleRef CFBundleGetBundleWithIdentifier (
   CFStringRef bundleID
);
```

Parameters

`bundleID`

The identifier of the bundle to locate. Note that identifier names are case-sensitive.

Return Value
A CFBundle object, or `NULL` if the bundle was not found. Ownership follows the Get Rule.

Discussion
For a bundle to be located using its identifier, the bundle object must have already been created. The principal intended purpose for locating bundles by identifier is so that code (in frameworks, plugins, etc.) can find its own bundle. If a bundle is created, then the bundle deleted from the filesystem and this function invoked afterwards, it will still return the original bundle.

Bundle identifiers are created by entering a value for the key `CFBundleIdentifier` in the bundle's `Info.plist` file.

To guarantee uniqueness, bundle identifiers take the form of reverse-DNS naming style package names, such as `com.MyCompany.MyApp.bundleName`.

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

Related Sample Code
BasicInputMethod
BetterAuthorizationSample
HID Utilities
QTPixelBufferVCToCGImage
SampleDS

Declared In
`CFBundle.h`

CFBundleGetDataPointerForName

Returns a data pointer to a symbol of the given name.

```
void * CFBundleGetDataPointerForName (
   CFBundleRef bundle,
   CFStringRef symbolName
);
```

Parameters

bundle

The bundle to examine.

symbolName

The name of the symbol you are searching for.

Return Value

A data pointer to a symbol named *symbolName* in *bundle*, or NULL if *symbolName* cannot be found. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFBundle.h

CFBundleGetDataPointersForNames

Returns a C array of data pointer to symbols of the given names.

```
void CFBundleGetDataPointersForNames (
   CFBundleRef bundle,
   CFArrayRef symbolNames,
   void *stbl[]
);
```

Parameters

bundle

The bundle to examine.

symbolNames

A CFArray object containing CFString objects representing the symbol names to search for.

stbl

A C array into which this function stores the data pointers for the symbols specified in *symbolNames*. The array contains NULL for any names in *symbolNames* that cannot be found.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFBundle.h

CFBundleGetDevelopmentRegion

Returns the bundle's development region from the bundle's information property list.

```
CFStringRef CFBundleGetDevelopmentRegion (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value
A CFString object containing the name of the bundle's development region. Ownership follows the Get Rule.

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

Declared In
CFBundle.h

CFBundleGetFunctionPointerForName

Returns a pointer to a function in a bundle's executable code using the function name as the search key.

```
void * CFBundleGetFunctionPointerForName (
   CFBundleRef bundle,
   CFStringRef functionName
);
```

Parameters

bundle

The bundle to examine.

functionName

The name of the function to locate.

Return Value
A pointer to a function in a *bundle*'s executable code, or NULL if *functionName* cannot be found. Ownership follows the Get Rule.

Discussion
Calling this function will cause the bundle's code to be loaded if necessary.

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

Related Sample Code
AuthForAll
MemoryBasedBundle
SpellingChecker CarbonCocoa Bundled
SpellingChecker-CarbonCocoa

Declared In
CFBundle.h

CFBundleGetFunctionPointersForNames

Constructs a function table containing pointers to all of the functions found in a bundle's main executable code.

```
void CFBundleGetFunctionPointersForNames (
   CFBundleRef bundle,
   CFArrayRef functionNames,
   void *ftbl[]
);
```

Parameters

bundle

The bundle to examine.

functionNames

A CFArray object containing a list of the function names to locate.

ftbl

A C array into which this function stores the function pointers for the symbols specified in *functionNames*. The array contains `NULL` for any names in *functionNames* that cannot be found.

Discussion

Calling this function causes the bundle's code to be loaded if necessary.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleGetIdentifier

Returns the bundle identifier from a bundle's information property list.

```
CFStringRef CFBundleGetIdentifier (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

A CFString object containing the bundle's identifier, or `NULL` if none was specified in the information property list. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BetterAuthorizationSample

CFPrefTopScores

Declared In

`CFBundle.h`

CFBundleGetInfoDictionary

Returns a bundle's information dictionary.

```
CFDictionaryRef CFBundleGetInfoDictionary (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

A CFDictionary object containing the data stored in the bundle's information property list (the `Info.plist` file). This is a global information dictionary. CFBundle may add extra keys to this dictionary for its own use. Ownership follows the Get Rule.

Discussion

You should typically use `CFBundleGetValueForInfoDictionaryKey` (page 130) rather than retrieving values directly from the info dictionary because the function will return localized values if any are available. Use `CFBundleGetInfoDictionary` only if you know that the key you are interested in will not be localized.

To retrieve an info dictionary without creating a CFBundle object, see `CFBundleCopyInfoDictionaryInDirectory` (page 110) and `CFBundleCopyInfoDictionaryForURL` (page 110).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

Declared In

`CFBundle.h`

CFBundleGetLocalInfoDictionary

Returns a bundle's localized information dictionary.

```
CFDictionaryRef CFBundleGetLocalInfoDictionary (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

A dictionary containing the key-value pairs in *bundle*'s localized information dictionary (from the `InfoPlist.strings` file for the current locale). Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

VertexPerformanceTest

Declared In
CFBundle.h

CFBundleGetMainBundle

Returns an application's main bundle.

```
CFBundleRef CFBundleGetMainBundle (
   void
);
```

Return Value
A CFBundle object representing the application's main bundle, or NULL if it is not possible to create a bundle. Ownership follows the Get Rule.

Discussion
CFBundle creates a main bundle whenever it possibly can, even for unbundled apps. There are a few situations in which it is not possible, so you should check the return value against NULL, but this happens only in exceptional circumstances.

For an explanation of the main bundle, see Locating and Opening Bundles in *Bundle Programming Guide*.

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

Related Sample Code
BetterAuthorizationSample
BSDLLCTest
CoreTextTest
FileNotification
HID Utilities Source

Declared In
CFBundle.h

CFBundleGetPackageInfo

Returns a bundle's package type and creator.

```
void CFBundleGetPackageInfo (
   CFBundleRef bundle,
   UInt32 *packageType,
   UInt32 *packageCreator
);
```

Parameters

bundle
The bundle to examine.

packageType
On return, the four-letter Mac OS-style type code for the bundle. This is APPL for applications, FMWK for frameworks, and BNDL for generic bundles. Or a more specific type code for generic bundles.

packageCreator
On return, the four-letter Mac OS-style "creator" code for the bundle.

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

Declared In
`CFBundle.h`

CFBundleGetPackageInfoInDirectory

Returns a bundle's package type and creator without having to create a CFBundle object.

```
Boolean CFBundleGetPackageInfoInDirectory (
   CFURLRef url,
   UInt32 *packageType,
   UInt32 *packageCreator
);
```

Parameters

url
The location of a bundle.

packageType
On return, the four-letter Mac OS-style type code for the bundle. This is `APPL` for applications, `FMWK` for frameworks, and `BNDL` for generic bundles. Or a more specific type code for generic bundles.

packageCreator
On return, the four-letter Mac OS-style "creator" code for the bundle.

Return Value
`true` if the package type and creator were found, otherwise `false`.

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

Declared In
`CFBundle.h`

CFBundleGetPlugIn

Returns a bundle's plug-in.

```
CFPlugInRef CFBundleGetPlugIn (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle to examine.

Return Value
The plug-in for *bundle*. Ownership follows the Get Rule.

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

Declared In
`CFBundle.h`

CFBundleGetTypeID

Returns the type identifier for the CFBundle opaque type.

```
CFTypeID CFBundleGetTypeID (
   void
);
```

Return Value
The type identifier for the CFBundle opaque type.

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

Declared In
`CFBundle.h`

CFBundleGetValueForInfoDictionaryKey

Returns a value (localized if possible) from a bundle's information dictionary.

```
CFTypeRef CFBundleGetValueForInfoDictionaryKey (
   CFBundleRef bundle,
   CFStringRef key
);
```

Parameters

bundle
The bundle to examine.

key
The key for the value to return.

Return Value
A value corresponding to *key* in *bundle*'s information dictionary. If available, a localized value is returned, otherwise the global value is returned. Ownership follows the Get Rule.

Discussion
You should use this function rather than retrieving values directly from the info dictionary (`Info.plist`) because `CFBundleGetValueForInfoDictionaryKey` returns localized values if any are available (from the `InfoPlist.strings` file for the current locale).

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

Declared In
`CFBundle.h`

CFBundleGetVersionNumber

Returns a bundle's version number.

```
UInt32 CFBundleGetVersionNumber (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine. The bundle's version number can be number or a string of the standard form "2.5.3d5".

Return Value

A Mac OS `vers` resource style version number. If the bundle's version number is a number, it is interpreted as the unsigned long integer format defined by the `vers` resource on Mac OS 9. If it is a string, it is automatically converted to the numeric representation, where the major version number is restricted to 2 BCD digits (in other words, it must be in the range 0-99) and the minor and bug fix version numbers are each restricted to a single BCD digit (0-9). See Technical Note TN1132 for more details.

Discussion

This function is only supported for the Mac OS `vers` resource style version numbers. Where other version number styles—namely X, or X.Y, or X.Y.Z—are used, you can use `CFBundleGetValueForInfoDictionaryKey` (page 130) with the key `kCFBundleVersionKey` to extract the version number as a string from the bundle's information dictionary.

Some version numbers of the form X, X.Y, and X.Y.Z may work with this function, if X <= 99, Y <= 9, and Z <= 9. Thus a version number 76.5.4 will work, but 76.12 will not work.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleIsExecutableLoaded

Obtains information about the load status for a bundle's main executable.

```
Boolean CFBundleIsExecutableLoaded (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle to examine.

Return Value

`true` if *bundle*'s main executable has been loaded, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFBundleLoadExecutable

Loads a bundle's main executable code into memory and dynamically links it into the running application.

```
Boolean CFBundleLoadExecutable (
   CFBundleRef bundle
);
```

Parameters

bundle

The bundle whose main executable you want to load.

Return Value

`true` if the executable was successfully loaded, otherwise `false`.

Discussion

You should typically try to avoid using this function, but instead use `CFBundleGetFunctionPointerForName` (page 125) and related functions since these make memory management of the bundle easier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AuthForAll

BSDLLCTest

MoreSCF

SpellingChecker CarbonCocoa Bundled

SpellingChecker-CarbonCocoa

Declared In

`CFBundle.h`

CFBundleLoadExecutableAndReturnError

Returns a Boolean value that indicates whether a given bundle is loaded, attempting to load it if necessary.

```
Boolean CFBundleLoadExecutableAndReturnError (
   CFBundleRef bundle,
   CFErrorRef *error
);
```

Parameters

bundle

The bundle to examine.

error

Upon return, if an error occurs contains a CFError that describes the problem. Ownership follows the Create Rule.

Return Value

If *bundle* is already loaded, returns `true`. If *bundle* is not already loaded, attempts to load *bundle*; if that attempt succeeds returns `true`, otherwise returns `false`.

Availability

Available in Mac OS X v10.5 and later.

Declared In
CFBundle.h

CFBundleOpenBundleResourceFiles

Opens the non-localized and localized resource files (if any) for a bundle in separate resource maps.

```
SInt32 CFBundleOpenBundleResourceFiles (
   CFBundleRef bundle,
   CFBundleRefNum *refNum,
   CFBundleRefNum *localizedRefNum
);
```

Parameters

bundle
The bundle whose resource map you want to open.

refNum
On return, the reference number of the non-localized resource map.

localizedRefNum
On return, the reference number of the localized resource map.

Return Value
An error code. The function returns 0 (noErr) if successful. If the bundle contains more than one resource file, the function returns an error code only if none was opened. The most common error is resFNotFound, but the function may also pass through other errors returned from the Resource Manager.

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

Declared In
CFBundle.h

CFBundleOpenBundleResourceMap

Opens the non-localized and localized resource files (if any) for a bundle in a single resource map.

```
CFBundleRefNum CFBundleOpenBundleResourceMap (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle whose resource map you want to open.

Return Value
A distinct reference number for the resource map.

Discussion
Creates and makes current a single read-only resource map containing the non-localized and localized resource files. If this function is called multiple times, it opens the files multiple times and returns distinct reference numbers for each. Use CFBundleCloseBundleResourceMap (page 106) to close a resource map.

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

Related Sample Code
SampleDS

Declared In
`CFBundle.h`

CFBundlePreflightExecutable

Returns a Boolean value that indicates whether a given bundle is loaded or appears to be loadable.

```
Boolean CFBundlePreflightExecutable (
   CFBundleRef bundle,
   CFErrorRef *error
);
```

Parameters

bundle
The bundle to examine.

error
Upon return, if an error occurs contains a CFError that describes the problem. Ownership follows the Create Rule.

Return Value
`true` if *bundle* is loaded or upon inspection appears to be loadable, otherwise `false`.

Discussion
If this function returns true, this does not mean that the bundle is definitively loadable, since it may fail to load due to link errors or other problems not readily detectable.

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

Declared In
`CFBundle.h`

CFBundleUnloadExecutable

Unloads the main executable for the specified bundle.

```
void CFBundleUnloadExecutable (
   CFBundleRef bundle
);
```

Parameters

bundle
The bundle whose main executable you want to unload.

Discussion
You should typically try to avoid using this function, but instead use `CFBundleGetFunctionPointerForName` (page 125) and related functions since these make management of the bundle easier (when the last reference to the CFBundle object is released, and it is finally deallocated, then the code will be unloaded if it is still loaded, and if the executable is of a type that supports unloading).

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

Related Sample Code
HID Utilities Source

Declared In
`CFBundle.h`

CFCopyLocalizedString

Searches the default strings file `Localizable.strings` for the string associated with the specified key.

```
CFStringRef CFCopyLocalizedString (
    CFStringRef key,
    const char *comment
);
```

Parameters

key

The development language version of the string. This string is used as the search key to locate the localized version of the string.

comment

A comment to provide the translators with contextual information necessary for proper translation.

Return Value
The localized version of the requested string. Returns *key* if no value corresponding to *key* is found. Ownership follows the Create Rule.

Discussion
This is a macro variant of `CFBundleCopyLocalizedString` (page 112) for use with the `genstrings` tool.

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

Related Sample Code
Cocoa PDE with Carbon Printing
MoreSCF
MovieVideoChart
Quartz2DBasics
SampleRaster

Declared In
`CFBundle.h`

CFCopyLocalizedStringFromTable

Searches the specified strings file for the string associated with the specified key.

```
CFStringRef CFCopyLocalizedStringFromTable (
    CFStringRef key,
    CFStringRef tableName,
    const char *comment
);
```

Parameters

key

The development language version of the string. This string is used as the search key to locate the localized version of the string.

tableName

The name of the strings file to search. The name should not include the `strings` filename extension. The case of the string must match that of the file name, even on file systems (such as HFS+) that are not case sensitive with regards to file names

comment

A comment to provide the translators with contextual information necessary for proper translation.

Return Value

The localized version of the requested string, or *key* if no value corresponding to *key* is found. Ownership follows the Create Rule.

Discussion

This is a macro variant of `CFBundleCopyLocalizedString` (page 112) for use with the `genstrings` tool.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFBundle.h`

CFCopyLocalizedStringFromTableInBundle

Returns a localized version of the specified string.

```
CFStringRef CFCopyLocalizedStringFromTableInBundle (
    CFStringRef key,
    CFStringRef tableName,
    CFBundleRef bundle,
    const char *comment
);
```

Parameters

key

The development language version of the string. This string is used as the search key to locate the localized version of the string.

tableName

The name of the strings file to search. The name should not include the `strings` filename extension. The case of the string must match that of the file name, even on file systems (such as HFS+) that are not case sensitive with regards to file names

bundle

The bundle to examine.

comment
A comment to provide the translators with contextual information necessary for proper translation.

Return Value
The localized version of the requested string, or *key* if no value corresponding to *key* is found. Ownership follows the Create Rule.

Discussion
This is a macro variant of `CFBundleCopyLocalizedString` (page 112) for use with the `genstrings` tool.

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

Related Sample Code
AudioCodecSDK
SampleAUs

Declared In
`CFBundle.h`

CFCopyLocalizedStringWithDefaultValue

Returns a localized version of a localization string.

```
CFStringRef CFCopyLocalizedStringWithDefaultValue (
    CFStringRef key,
    CFStringRef tableName,
    CFBundleRef bundle,
    CFStringRef value,
    const char *comment
);
```

Parameters

key
The development language version of the string. This string is used as the search key to locate the localized version of the string.

tableName
The name of the strings file to search. The name should not include the `strings` filename extension.

bundle
The bundle to examine.

value
The default value for the requested localization string.

comment
A comment to provide the translators with contextual information necessary for proper translation.

Return Value
The localized version of the requested string, or *key* if no value corresponding to *key* is found. Ownership follows the Create Rule.

Discussion
This is a macro variant of `CFBundleCopyLocalizedString` (page 112) for use with the `genstrings` tool.

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

Declared In
`CFBundle.h`

Data Types

CFBundleRef

A reference to a CFBundle object.

```
typedef struct __CFBundle *CFBundleRef;
```

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

Declared In
`CFBundle.h`

CFBundleRefNum

Type that identifies a distinct reference number for a resource map.

```
#if __LP64__
typedef int CFBundleRefNum;
#else /* __LP64__ */
typedef SInt16 CFBundleRefNum;
#endif /* __LP64__ */
```

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

Declared In
`CFBundle.h`

Constants

Information Property List Keys

Standard keys found in a bundle's information property list file.

```
const CFStringRef kCFBundleInfoDictionaryVersionKey;
const CFStringRef kCFBundleExecutableKey;
const CFStringRef kCFBundleIdentifierKey;
const CFStringRef kCFBundleVersionKey;
const CFStringRef kCFBundleDevelopmentRegionKey;
const CFStringRef kCFBundleNameKey;
const CFStringRef kCFBundleLocalizationsKey;
```

Constants

`kCFBundleInfoDictionaryVersionKey`
The version of the information property list format.

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleExecutableKey`
The name of the executable in this bundle (if any).

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleIdentifierKey`
The bundle identifier.

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleVersionKey`
The version number of the bundle.

For Mac OS 9 style version numbers (for example "2.5.3d5"), clients can use `CFBundleGetVersionNumber` (page 130) instead of accessing this key directly since that function will properly convert the version string into its compact integer representation.

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleDevelopmentRegionKey`
The name of the development language of the bundle.

When CFBundle looks for resources, the fallback is to look in the lproj whose name is given by the `kCFBundleDevelopmentRegionKey` in the `Info.plist` file. You must, therefore, ensure that a bundle contains an lproj with that exact name containing a copy of every localized resource, otherwise CFBundle cannot guarantee the fallback mechanism will work.

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleNameKey`
The human-readable name of the bundle.

This key is often found in the `InfoPlist.strings` since it is usually localized.

Available in Mac OS X v10.0 and later.

Declared in `CFBundle.h`.

`kCFBundleLocalizationsKey`
Allows an unbundled application that handles localization itself to specify which localizations it has available.

Available in Mac OS X v10.2 and later.

Declared in `CFBundle.h`.

Declared In
CFBundle.h

Architecture Types

Constants that identify executable architecture types.

```
enum {
    kCFBundleExecutableArchitectureI386     = 0x00000007,
    kCFBundleExecutableArchitecturePPC      = 0x00000012,
    kCFBundleExecutableArchitectureX86_64   = 0x01000007,
    kCFBundleExecutableArchitecturePPC64    = 0x01000012
};
```

Constants

`kCFBundleExecutableArchitectureI386`
Specifies the 32-bit Intel architecture.

Available in Mac OS X v10.5 and later.

Declared in `CFBundle.h`.

`kCFBundleExecutableArchitecturePPC`
Specifies the 32-bit PowerPC architecture.

Available in Mac OS X v10.5 and later.

Declared in `CFBundle.h`.

`kCFBundleExecutableArchitectureX86_64`
Specifies the 64-bit Intel architecture.

Available in Mac OS X v10.5 and later.

Declared in `CFBundle.h`.

`kCFBundleExecutableArchitecturePPC64`
Specifies the 64-bit PowerPC architecture.

Available in Mac OS X v10.5 and later.

Declared in `CFBundle.h`.

Declared In
CFBundle.h

CFCalendar Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFCalendar.h
Companion guides	Locales Programming Guide Date and Time Programming Guide for Core Foundation Internationalization Programming Topics

Overview

The CFCalendar opaque type represents a calendar system. The associated API provides information about a calendar and supports calendrical computations such as determining the range of a given calendrical unit and adding units to a given absolute time.

CFAbsoluteTime is the operational lingua franca of CFCalendar—to do calendar arithmetic, you start and end with an absolute time; to convert between a decomposed date in one calendar and another calendar, you first convert to an absolute time. CFAbsoluteTime provides the absolute scale and epoch for dates and times, which can then be rendered into a particular calendar, for calendrical computations or user display.

In a calendar, day, week, weekday, month, and year numbers are generally 1-based, but there may be calendar-specific exceptions. Ordinal numbers, where they occur, are 1-based. Some calendars represented by this API may have to map their basic unit concepts into year/month/week/day/… nomenclature. For example, a calendar composed of 4 quarters in a year instead of 12 months uses the "month" unit to represent quarters. The particular values of the unit are defined by each calendar, and are not necessarily "consistent with" or have a "correspondence with," values for that unit in another calendar.

Several CFCalendar functions (`CFCalendarComposeAbsoluteTime` (page 145), `CFCalendarDecomposeAbsoluteTime` (page 148), `CFCalendarAddComponents` (page 144), and `CFCalendarGetComponentDifference` (page 149)) take a description string that describes the calendrical components provided in a varargs parameter area. You can provide as many components as you need (or choose to), in whatever order you choose. When there is incomplete information to compute an absolute time, default values similar to 0 and 1 are usually chosen by a calendar, but this is a calendar-specific choice. If you provide inconsistent information, calendar-specific disambiguation is performed (which may involve ignoring one or more of the parameters). The characters of the description string specify the units and order of the parameters which follow. The characters are adopted from the corresponding format characters used by CFDateFormatter when possible, as shown in Table 9-1.

Table 9-1 Calendrical components parameter descriptors

Symbol	Meaning	Value Type
y	year	int
M	month	int
d	day	int
H	hour	int
m	minute	int
s	second	int

Information related to formatting dates and times and name-related calendar information is managed by CFDateFormatter.

CFCalendar is subject to some limitations. There is no leap second handling—the existence of leap seconds is ignored as in the other CoreFoundation API. In general, historical accuracy of calendars is not guaranteed. There is currently no API for defining your own calendars.

CFCalendar is "toll-free bridged" with its Cocoa Foundation counterpart, NSCalendar. 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 `NSCalendar *` parameter, you can pass in a `CFCalendarRef`, and in a function where you see a `CFCalendarRef` parameter, you can pass in an NSCalendar instance. See Interchangeable Data Types for more information on toll-free bridging.

The CFCalendar opaque type is available in Mac OS X v10.4 and later.

Functions by Task

Creating a Calendar

`CFCalendarCopyCurrent` (page 146)
Returns a copy of the logical calendar for the current user.

`CFCalendarCreateWithIdentifier` (page 147)
Returns a calendar object for the calendar identified by a calendar identifier.

Calendrical Calculations

`CFCalendarAddComponents` (page 144)
Computes the absolute time when specified components are added to a given absolute time.

`CFCalendarComposeAbsoluteTime` (page 145)
Computes the absolute time from components in a description string.

`CFCalendarDecomposeAbsoluteTime` (page 148)
Computes the components which are indicated by the componentDesc description string for the given absolute time.

`CFCalendarGetComponentDifference` (page 149)
Computes the difference between the two absolute times, in terms of specified calendrical components.

Getting Ranges of Units

`CFCalendarGetRangeOfUnit` (page 153)
Returns the range of values that one unit can take on within a larger unit during which a specific absolute time occurs.

`CFCalendarGetOrdinalityOfUnit` (page 152)
Returns the ordinal number of a calendrical unit within a larger unit at a specified absolute time.

`CFCalendarGetTimeRangeOfUnit` (page 153)
Returns by reference the start time and duration of a given calendar unit that contains a given absolute time.

`CFCalendarGetMaximumRangeOfUnit` (page 150)
Returns the maximum range limits of the values that a specified unit can take on in a given calendar.

`CFCalendarGetMinimumRangeOfUnit` (page 151)
Returns the minimum range limits of the values that a specified unit can take on in a given calendar.

Getting and Setting the Time Zone

`CFCalendarCopyTimeZone` (page 147)
Returns a time zone object for a specified calendar.

`CFCalendarSetTimeZone` (page 155)
Sets the time zone for a calendar.

Getting the Identifier

`CFCalendarGetIdentifier` (page 150)
Returns the given calendar's identifier.

Getting and Setting the Locale

`CFCalendarCopyLocale` (page 146)
Returns a locale object for a specified calendar.

`CFCalendarSetLocale` (page 155)
Sets the locale for a calendar.

Getting and Setting Day Information

`CFCalendarGetFirstWeekday` (page 150)
Returns the index of first weekday for a specified calendar.

`CFCalendarSetFirstWeekday` (page 154)
Sets the first weekday for a calendar.

`CFCalendarGetMinimumDaysInFirstWeek` (page 151)
Returns the minimum number of days in the first week of a specified calendar.

`CFCalendarSetMinimumDaysInFirstWeek` (page 155)
Sets the minimum number of days in the first week of a specified calendar.

Getting the Type ID

`CFCalendarGetTypeID` (page 154)
Returns the type identifier for the CFCalendar opaque type.

Functions

CFCalendarAddComponents

Computes the absolute time when specified components are added to a given absolute time.

```
Boolean CFCalendarAddComponents (
   CFCalendarRef calendar,
   CFAbsoluteTime *at,
   CFOptionFlags options,
   const unsigned char *componentDesc,
   ...
);
```

Parameters

calendar
The calendar to use for the computation.

at
A reference to an absolute time. On input, points to the absolute time to which components are to be added; on output, points to the result of the computation.

options
Options for the calculation. For valid values, see "Constants" (page 156).

componentDesc
A string that describes the components provided in the vararg parameters.

...
Vararg parameters giving amounts of each calendrical component in the order specified by *componentDesc*. The amounts to add may be negative, zero, positive, or any combination thereof.

Return Value
TRUE—and in *at* the computed time—if *at* falls inside the defined range of the calendar and it is possible to calculate the absolute time when the components (the calendrical components specified by *componentDesc* and given in the varargs) are added to the input absolute time *at*; otherwise FALSE.

Discussion
Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally components are added in the order specified.

If you specify a "wrap" option (kCFCalendarComponentsWrap (page 158)), the specified components should be incremented and wrap around to zero/one on overflow, but should not cause higher units to be incremented. When "Wrap" is false, overflow in a unit carries into the higher units, as in typical addition.

Note that some computations can take a relatively long time to perform.

The following example shows how to add 2 months and 3 days to absolute time at's current value using an existing calendar (gregorian):

```
CFCalendarAddComponents(gregorian, &at, 0, "Md",  2, 3);
```

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

Declared In
CFCalendar.h

CFCalendarComposeAbsoluteTime

Computes the absolute time from components in a description string.

```
Boolean CFCalendarComposeAbsoluteTime (
   CFCalendarRef calendar,
   CFAbsoluteTime *at,
   const unsigned char *componentDesc,
   ...
);
```

Parameters

calendar
The calendar to use for the computation.

at
Upon return, contains the computed absolute time.

componentDesc
A string that describes the components provided in the vararg parameters.

...
Vararg parameters giving amounts of each calendrical component in the order specified by *componentDesc*. The amounts to add may be negative, zero, positive, or any combination thereof.

Return Value
TRUE—and in *at* the absolute time computed from the given components—if the *componentDesc* description string can be converted into an absolute time, otherwise FALSE. Also returns FALSE for out-of-range values.

Discussion
When there are insufficient components provided to completely specify an absolute time, a calendar uses default values of its choice. When there is inconsistent information, a calendar may ignore some of the parameters or the function may return `FALSE`. Unnecessary components are ignored (for example, Day takes precedence over Weekday + Weekday ordinal). Note that some computations can take a relatively long time to perform.

The following example shows how to use this function to initialize an absolute time, `at`, to 6 January 1965 14:10:00, for a given calendar `gregorian`.

```
CFCalendarComposeAbsoluteTime(gregorian, &at, "yMdHms",  1965, 1, 6, 14, 10,
00);
```

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

Declared In
`CFCalendar.h`

CFCalendarCopyCurrent

Returns a copy of the logical calendar for the current user.

```
CFCalendarRef CFCalendarCopyCurrent (
   void
);
```

Return Value
The logical calendar for the current user that is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences. This function may return a retained cached object, not a new object. Ownership follows the Create Rule.

Discussion
Settings you get from this calendar do not change if user defaults change so that your operations are consistent.

Typically you perform some operations on the returned object and then release it. The returned object may be cached, so you do not need to hold on to it indefinitely.

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

Declared In
`CFCalendar.h`

CFCalendarCopyLocale

Returns a locale object for a specified calendar.

```
CFLocaleRef CFCalendarCopyLocale (
   CFCalendarRef calendar
);
```

Parameters

calendar

The calendar to examine.

Return Value

A copy of the locale object for the specified calendar. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.4 and later.

Declared In

CFCalendar.h

CFCalendarCopyTimeZone

Returns a time zone object for a specified calendar.

```
CFTimeZoneRef CFCalendarCopyTimeZone (
   CFCalendarRef calendar
);
```

Parameters

calendar

The calendar to examine.

Return Value

A copy of the time zone object for the specified calendar. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.4 and later.

Declared In

CFCalendar.h

CFCalendarCreateWithIdentifier

Returns a calendar object for the calendar identified by a calendar identifier.

```
CFCalendarRef CFCalendarCreateWithIdentifier (
   CFAllocatorRef allocator,
   CFStringRef identifier
);
```

Parameters

alloc

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

ident

A calendar identifier. Calendar identifier constants are given in `CFLocaleRef` (page 252).

Return Value
A calendar object for the calendar identified by *ident*. If the identifier is unknown (if, for example, it is either an unrecognized string, or the calendar is not supported by the current version of the operating system), returns `NULL`. Ownership follows the Create Rule.

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

Declared In
`CFCalendar.h`

CFCalendarDecomposeAbsoluteTime

Computes the components which are indicated by the componentDesc description string for the given absolute time.

```
Boolean CFCalendarDecomposeAbsoluteTime (
   CFCalendarRef calendar,
   CFAbsoluteTime at,
   const unsigned char *componentDesc,
   ...
);
```

Parameters

calendar
The calendar to use for the computation.

at
An absolute time.

componentDesc
A string that describes the components provided in the vararg parameters.

...
Vararg pointers to storage for each of the desired components. On successful return, the pointers are filled with values of the corresponding components. The type of all units is `int`.

Return Value
`TRUE` if the function is able to compute the components indicated by the *componentDesc* description string for the given absolute time, and fills the values to the components given in the varargs. Returns `FALSE` if the absolute time falls outside the defined range of the calendar, or the computation cannot be performed.

Discussion
The Weekday ordinality, when requested, refers to the next larger (than Week) of the requested units. Some computations can take a relatively long time to perform.

The following example shows how to use this function to determine the current year, month, and day, using an existing calendar (`gregorian`):

```
CFCalendarDecomposeAbsoluteTime(gregorian, CFAbsoluteTimeGetCurrent(), "yMd",
 &year, &month, &day);
```

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

Declared In
`CFCalendar.h`

CFCalendarGetComponentDifference

Computes the difference between the two absolute times, in terms of specified calendrical components.

```
Boolean CFCalendarGetComponentDifference (
   CFCalendarRef calendar,
   CFAbsoluteTime startingAT,
   CFAbsoluteTime resultAT,
   CFOptionFlags options,
   const unsigned char *componentDesc,
   ...
);
```

Parameters

calendar
The calendar to use for the computation.

startingAT
The starting absolute time.

resultAT
The result absolute time.

options
Options for the calculation. For valid values, see "Constants" (page 156).

componentDesc
A string that describes the components provided in the vararg parameters.

...
Vararg pointers to storage for each of the desired components. On successful return, the pointers are filled with values of the corresponding components. The type of all units is `int`.

Return Value

`TRUE`—and in the varargs the differences—if it is possible to calculate the difference (result - starting) between *resultAT* and *startingAT* in terms of the calendrical components specified by *componentDesc*. Returns `FALSE` if either absolute time falls outside the defined range of the calendar, or the computation cannot be performed.

Discussion

The result is lossy if there isn't a small enough unit requested to hold the full precision of the difference. Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally larger components will be computed before smaller components; for example, in the Gregorian calendar a result might be `1` month and `5` days, instead of, for example, `0` months and `35` days. The resulting component values may be negative if later is before earlier.

This computation is roughly the inverse of the `CFCalendarAddComponents` (page 144) operation, but calendrical arithmetic is invertible only in simple cases. This computation tends to be several times more expensive than the Add operation.

The following example shows how to get the approximate number of days between two absolute times (`at1`, `at2`) using an existing calendar (`gregorian`):

```
CFCalendarGetComponentDifference(gregorian, at1, at2, 0, "d",  &days);
```

Availability

Available in Mac OS X v10.4 and later.

Declared In
CFCalendar.h

CFCalendarGetFirstWeekday

Returns the index of first weekday for a specified calendar.

```
CFIndex CFCalendarGetFirstWeekday (
   CFCalendarRef calendar
);
```

Parameters

calendar
The calendar to examine.

Return Value
The index of the first weekday of the specified calendar.

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

Declared In
CFCalendar.h

CFCalendarGetIdentifier

Returns the given calendar's identifier.

```
CFStringRef CFCalendarGetIdentifier (
   CFCalendarRef calendar
);
```

Parameters

calendar
The calendar to examine.

Return Value
A string representation of *calendar*'s identifier. Calendar identifier constants can be found in CFLocaleRef (page 252). Ownership follows the Get Rule.

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

Declared In
CFCalendar.h

CFCalendarGetMaximumRangeOfUnit

Returns the maximum range limits of the values that a specified unit can take on in a given calendar.

```
CFRange CFCalendarGetMaximumRangeOfUnit (
   CFCalendarRef calendar,
   CFCalendarUnit unit
);
```

Parameters

calendar

The calendar to examine.

unit

A calendar unit. For valid values see `CFCalendarUnit` (page 156).

Return Value

The maximum range limits of the values that the specified unit can take on in *calendar*. For example, in the Gregorian calendar the maximum ranges for the Day unit is 1-31.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFCalendar.h`

CFCalendarGetMinimumDaysInFirstWeek

Returns the minimum number of days in the first week of a specified calendar.

```
CFIndex CFCalendarGetMinimumDaysInFirstWeek (
   CFCalendarRef calendar
);
```

Parameters

calendar

The calendar to examine.

Return Value

The minimum number of days in the first week of *calendar*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFCalendar.h`

CFCalendarGetMinimumRangeOfUnit

Returns the minimum range limits of the values that a specified unit can take on in a given calendar.

```
CFRange CFCalendarGetMinimumRangeOfUnit (
   CFCalendarRef calendar,
   CFCalendarUnit unit
);
```

Parameters

calendar

The calendar to examine.

unit
A calendar unit. For valid values see `CFCalendarUnit` (page 156).

Return Value
The minimum range limits of the values that the specified unit can take on in *calendar*. For example, in the Gregorian calendar the minimum ranges for the Day unit is 1-28.

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

Declared In
`CFCalendar.h`

CFCalendarGetOrdinalityOfUnit

Returns the ordinal number of a calendrical unit within a larger unit at a specified absolute time.

```
CFIndex CFCalendarGetOrdinalityOfUnit (
   CFCalendarRef calendar,
   CFCalendarUnit smallerUnit,
   CFCalendarUnit biggerUnit,
   CFAbsoluteTime at
);
```

Parameters
calendar
The calendar to examine.

smallerUnit
A calendar unit. For valid values see `CFCalendarUnit` (page 156).

biggerUnit
A calendar unit. For valid values see `CFCalendarUnit` (page 156).

at
An absolute time.

Return Value
The ordinal number of the calendar unit specified by *smallerUnit* within the calendar unit specified by *biggerUnit* at the absolute time *at*. For example, the time 00:45 is in the first hour of the day, and for units Hour and Day respectively, the result would be 1.

If the *biggerUnit* parameter is not logically bigger than the *smallerUnit* parameter in the calendar, or the given combination of units does not make sense (or is a computation which is undefined), the result is `kCFNotFound`.

Discussion
The ordinality is in most cases not the same as the decomposed value of the unit. Typically return values are `1` and greater; an exception is the week-in-month calculation, which returns `0` for days before the first week in the month containing the date. Note that some computations can take a relatively long time to perform.

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

Declared In
`CFCalendar.h`

CFCalendarGetRangeOfUnit

Returns the range of values that one unit can take on within a larger unit during which a specific absolute time occurs.

```
CFRange CFCalendarGetRangeOfUnit (
   CFCalendarRef calendar,
   CFCalendarUnit smallerUnit,
   CFCalendarUnit biggerUnit,
   CFAbsoluteTime at
);
```

Parameters

calendar
The calendar to examine.

smallerUnit
A calendar unit. For valid values see `CFCalendarUnit` (page 156).

biggerUnit
A calendar unit. For valid values see `CFCalendarUnit` (page 156).

at
An absolute time.

Return Value

The range of values that the calendar unit specified by *smallerUnit* can take on within the calendar unit specified by *biggerUnit* that includes the absolute time *at*. For example, the range the Day unit can take on in the Month in which the absolute time lies.

If *biggerUnit* is not logically bigger than *smallerUnit* in the calendar, or the given combination of units does not make sense (or is a computation which is undefined), the result is `{kCFNotFound, kCFNotFound}`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFCalendar.h`

CFCalendarGetTimeRangeOfUnit

Returns by reference the start time and duration of a given calendar unit that contains a given absolute time.

```
Boolean CFCalendarGetTimeRangeOfUnit (
    CFCalendarRef calendar,
    CFCalendarUnit unit,
    CFAbsoluteTime at,
    CFAbsoluteTime *startp,
    CFTimeInterval *tip
);
```

Parameters

calendar
The calendar to examine.

unit
A calendar unit (for valid values, see `CFCalendarUnit` (page 156)).

at
An absolute time.

startp
Upon return, contains the beginning of the calendar unit specified by *unit* that contains the time *at*.

tip
Upon return, contains the duration of the calendar unit specified by *unit* that contains the time *at*.

Return Value
`true` if the values of *startp* and *tip* could be calculated, otherwise `false`.

Discussion
The function may fail if, for example, you try to get the range of a `kCFCalendarUnitWeekday` and specify a time (*at*) that is during a weekend.

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

Declared In
`CFCalendar.h`

CFCalendarGetTypeID

Returns the type identifier for the CFCalendar opaque type.

```
CFTypeID CFCalendarGetTypeID (
   void
);
```

Return Value
The type identifier for the CFCalendar opaque type.

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

Declared In
`CFCalendar.h`

CFCalendarSetFirstWeekday

Sets the first weekday for a calendar.

```
void CFCalendarSetFirstWeekday (
   CFCalendarRef calendar,
   CFIndex wkdy
);
```

Parameters

calendar
The calendar to modify.

wkdy
The index to set for the first weekday of *calendar*.

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

Declared In
`CFCalendar.h`

CFCalendarSetLocale

Sets the locale for a calendar.

```
void CFCalendarSetLocale (
   CFCalendarRef calendar,
   CFLocaleRef locale
);
```

Parameters

calendar
The calendar to modify.

locale
The locale to set for *calendar*.

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

Declared In
`CFCalendar.h`

CFCalendarSetMinimumDaysInFirstWeek

Sets the minimum number of days in the first week of a specified calendar.

```
void CFCalendarSetMinimumDaysInFirstWeek (
   CFCalendarRef calendar,
   CFIndex mwd
);
```

Parameters

calendar
The calendar to modify.

mwd
The number to set as the minimum number of days in the first week of *calendar*.

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

Declared In
`CFCalendar.h`

CFCalendarSetTimeZone

Sets the time zone for a calendar.

```
void CFCalendarSetTimeZone (
   CFCalendarRef calendar,
   CFTimeZoneRef tz
);
```

Parameters

calendar

The calendar to modify.

locale

The time zone to set for *calendar*.

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

Declared In
CFCalendar.h

Data Types

CFCalendarRef

A reference to a CFCalendar object.

```
typedef const struct __CFCalendar *CFCalendarRef;
```

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

Declared In
CFCalendar.h

Constants

CFCalendarUnit

CFCalendarUnit constants are used to specify calendrical units, such as day or month, in various calendar calculations.

```
typedef enum {
    kCFCalendarUnitEra = (1 << 1),
    kCFCalendarUnitYear = (1 << 2),
    kCFCalendarUnitMonth = (1 << 3),
    kCFCalendarUnitDay = (1 << 4),
    kCFCalendarUnitHour = (1 << 5),
    kCFCalendarUnitMinute = (1 << 6),
    kCFCalendarUnitSecond = (1 << 7),
    kCFCalendarUnitWeek = (1 << 8),
    kCFCalendarUnitWeekday = (1 << 9),
    kCFCalendarUnitWeekdayOrdinal = (1 << 10),
} CFCalendarUnit;
```

Constants

`kCFCalendarUnitEra`
Specifies the era unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitYear`
Specifies the year unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitMonth`
Specifies the month unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitDay`
Specifies the day unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitHour`
Specifies the hour unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitMinute`
Specifies the minute unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitSecond`
Specifies the second unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitWeek`
Specifies the week unit.
Available in Mac OS X v10.4 and later.
Declared in `CFCalendar.h`.

`kCFCalendarUnitWeekday`
Specifies the weekday unit.

The weekday units are the numbers `1-N` (where for the Gregorian calendar `N=7` and `1` is Sunday).

Available in Mac OS X v10.4 and later.

Declared in `CFCalendar.h`.

`kCFCalendarUnitWeekdayOrdinal`
Specifies the ordinal weekday unit.

The weekday ordinal unit describes ordinal position within the month unit of the corresponding weekday unit. For example, in the Gregorian calendar a weekday ordinal unit of `2` for a weekday unit `3` indicates "the second Tuesday in the month".

Available in Mac OS X v10.4 and later.

Declared in `CFCalendar.h`.

Component Wrapping Options

The wrapping option specifies overflow behavior for calendar components in calendrical calculations—see `CFCalendarAddComponents` (page 144) and `CFCalendarGetComponentDifference` (page 149).

```
enum {
    kCFCalendarComponentsWrap = (1 << 0)
}
```

Constants

`kCFCalendarComponentsWrap`
Specifies that the components specified for calendar components should be incremented and wrap around to zero/one on overflow, but should not cause higher units to be incremented.

Available in Mac OS X v10.4 and later.

Declared in `CFCalendar.h`.

CFCharacterSet Reference

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

Overview

A CFCharacterSet object represents a set of Unicode compliant characters. CFString uses CFCharacterSet objects to group characters together for searching operations, so that they can find any of a particular set of characters during a search. The two opaque types, CFCharacterSet and CFMutableCharacterSet, define the interface for static and dynamic character sets, respectively. The objects you create using these opaque types are referred to as character set objects (and when no confusion will result, merely as character sets).

CFCharacterSet's principal function, `CFCharacterSetIsCharacterMember` (page 165), provides the basis for all other functions in its interface. You create a character set using one of the `CFCharacterSetCreate...` functions. You may also use any one of the predefined character sets using the `CFCharacterSetGetPredefined` (page 164) function.

CFCharacterSet is "toll-free bridged" with its Cocoa Foundation counterpart, NSCharacterSet. 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 `NSCharacterSet *` parameter, you can pass in a `CFCharacterSetRef`, and in a function where you see a `CFCharacterSetRef` parameter, you can pass in an NSCharacterSet instance. This capability also applies to concrete subclasses of NSCharacterSet. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating Character Sets

`CFCharacterSetCreateCopy` (page 161)
Creates a new character set with the values from a given character set.

`CFCharacterSetCreateInvertedSet` (page 161)
Creates a new immutable character set that is the invert of the specified character set.

`CFCharacterSetCreateWithCharactersInRange` (page 163)
Creates a new character set with the values from the given range of Unicode characters.

`CFCharacterSetCreateWithCharactersInString` (page 163)
Creates a new character set with the values in the given string.

`CFCharacterSetCreateWithBitmapRepresentation` (page 162)
Creates a new immutable character set with the bitmap representation specified by given data.

Getting Predefined Character Sets

`CFCharacterSetGetPredefined` (page 164)
Returns a predefined character set.

Querying Character Sets

`CFCharacterSetCreateBitmapRepresentation` (page 160)
Creates a new immutable data with the bitmap representation from the given character set.

`CFCharacterSetHasMemberInPlane` (page 164)
Reports whether or not a character set contains at least one member character in the specified plane.

`CFCharacterSetIsCharacterMember` (page 165)
Reports whether or not a given Unicode character is in a character set.

`CFCharacterSetIsLongCharacterMember` (page 165)
Reports whether or not a given UTF-32 character is in a character set.

`CFCharacterSetIsSupersetOfSet` (page 166)
Reports whether or not a character set is a superset of another set.

Getting the Character Set Type Identifier

`CFCharacterSetGetTypeID` (page 164)
Returns the type identifier of the CFCharacterSet opaque type.

Functions

CFCharacterSetCreateBitmapRepresentation

Creates a new immutable data with the bitmap representation from the given character set.

```
CFDataRef CFCharacterSetCreateBitmapRepresentation (
   CFAllocatorRef alloc,
   CFCharacterSetRef theSet
);
```

Parameters

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

theSet

The set from which to create a bitmap representation. Refer to the comments for `CFCharacterSetCreateWithBitmapRepresentation` (page 162) for the detailed discussion of the bitmap representation format.

Return Value
A new CFData object containing a bitmap representation of *theSet*. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetCreateCopy

Creates a new character set with the values from a given character set.

```
CFCharacterSetRef CFCharacterSetCreateCopy (
   CFAllocatorRef alloc,
   CFCharacterSetRef theSet
);
```

Parameters

alloc

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

theSet

The character set to copy.

Return Value
A new character set that is a copy of *theSet*. Ownership follows the Create Rule.

Discussion
This function tries to compact the backing store where applicable.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetCreateInvertedSet

Creates a new immutable character set that is the invert of the specified character set.

```
CFCharacterSetRef CFCharacterSetCreateInvertedSet (
   CFAllocatorRef alloc,
   CFCharacterSetRef theSet
);
```

Parameters

alloc

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

theSet

The character set from which to create an inverted set.

Return Value

A new character set that is the invert of *theSet*. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetCreateWithBitmapRepresentation

Creates a new immutable character set with the bitmap representation specified by given data.

```
CFCharacterSetRef CFCharacterSetCreateWithBitmapRepresentation (
   CFAllocatorRef alloc,
   CFDataRef theData
);
```

Parameters

alloc

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

theData

A CFData object that specifies the bitmap representation of the Unicode character points the for the new character set. The bitmap representation could contain all the Unicode character range starting from BMP to Plane 16. The first 8KiB (8192 bytes) of the data represent the BMP range. The BMP range 8KiB can be followed by zero to sixteen 8KiB bitmaps, each prepended with the plane index byte. For example, the bitmap representing the BMP and Plane 2 has the size of 16385 bytes (8KiB for BMP, 1 byte index, and a 8KiB bitmap for Plane 2). The plane index byte, in this case, contains the integer value two.

If the data contains a Plane index byte outside of the valid Plane range (1 to 16), the behavior is undefined.

Return Value

A new character set containing the indicated characters from *theData*. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetCreateWithCharactersInRange

Creates a new character set with the values from the given range of Unicode characters.

```
CFCharacterSetRef CFCharacterSetCreateWithCharactersInRange (
   CFAllocatorRef alloc,
   CFRange theRange
);
```

Parameters

alloc

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

theRange

The Unicode range of characters of the new character set. The function accepts the range in 32-bit in the UTF-32 format. The valid character point range is from 0x00000 to 0x10FFFF.

Return Value

A new character set that contains a contiguous range of Unicode characters. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetCreateWithCharactersInString

Creates a new character set with the values in the given string.

```
CFCharacterSetRef CFCharacterSetCreateWithCharactersInString (
   CFAllocatorRef alloc,
   CFStringRef theString
);
```

Parameters

alloc

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

theString

A string containing the characters for the new set.

Return Value

A new character set containing the characters from *theString*. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetGetPredefined

Returns a predefined character set.

```
CFCharacterSetRef CFCharacterSetGetPredefined (
   CFCharacterSetPredefinedSet theSetIdentifier
);
```

Parameters

theSetIdentifier

A predefined character set. See "Predefined CFCharacterSet Selector Values" (page 167) for the list of available character sets.

Return Value

A predefined character set. This instance is owned by Core Foundation.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFCharacterSet.h

CFCharacterSetGetTypeID

Returns the type identifier of the CFCharacterSet opaque type.

```
CFTypeID CFCharacterSetGetTypeID (
   void
);
```

Return Value

The type identifier of the CFCharacterSet opaque type.

Discussion

CFMutableCharacterSet objects have the same type identifier as CFCharacterSet objects.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFCharacterSet.h

CFCharacterSetHasMemberInPlane

Reports whether or not a character set contains at least one member character in the specified plane.

```
Boolean CFCharacterSetHasMemberInPlane (
   CFCharacterSetRef theSet,
   CFIndex thePlane
);
```

Parameters

theSet

The character set to examine.

thePlane

The plane number to be checked for the membership. The valid value range is from 0 to 16. If the value is outside of the valid plane number range, the behavior is undefined.

Return Value

`true` if at least one member character is in the specified plane, otherwise `false`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetIsCharacterMember

Reports whether or not a given Unicode character is in a character set.

```
Boolean CFCharacterSetIsCharacterMember (
   CFCharacterSetRef theSet,
   UniChar theChar
);
```

Parameters

theSet

The character set to examine.

theChar

The Unicode character for which to test against the character set. Note that this function takes 16-bit Unicode character value; hence, it does not support access to the non-BMP planes.

Return Value

`true` if *theSet* contains *theChar*, otherwise `false`.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetIsLongCharacterMember

Reports whether or not a given UTF-32 character is in a character set.

```
Boolean CFCharacterSetIsLongCharacterMember (
   CFCharacterSetRef theSet,
   UTF32Char theChar
);
```

Parameters

theSet

The character set to examine.

theChar

The UTF-32 character for which to test against the character set.

Return Value

`true` if *theSet* contains *theChar*, otherwise `false`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetIsSupersetOfSet

Reports whether or not a character set is a superset of another set.

```
Boolean CFCharacterSetIsSupersetOfSet (
   CFCharacterSetRef theSet,
   CFCharacterSetRef theOtherset
);
```

Parameters

theSet

The character set to be checked for the membership of *theOtherSet*.

theOtherSet

The character set to be checked whether or not it is a subset of *theSet*.

Return Value

`true` if *theSet* is a superset of *theOtherSet*, otherwise `false`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFCharacterSet.h`

Data Types

CFCharacterSetPredefinedSet

Defines a predefined character set.

```
typedef CFIndex CFCharacterSetPredefinedSet;
```

Discussion
See "Predefined CFCharacterSet Selector Values" (page 167) for values.

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

Declared In
`CFCharacterSet.h`

CFCharacterSetRef

A reference to an immutable character set object.

```
typedef const struct __CFCharacterSet *CFCharacterSetRef;
```

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

Declared In
`CFCharacterSet.h`

Constants

Predefined CFCharacterSet Selector Values

Identifiers for the available predefined CFCharacterSet objects.

```
enum {
    kCFCharacterSetControl = 1,
    kCFCharacterSetWhitespace,
    kCFCharacterSetWhitespaceAndNewline,
    kCFCharacterSetDecimalDigit,
    kCFCharacterSetLetter,
    kCFCharacterSetLowercaseLetter,
    kCFCharacterSetUppercaseLetter,
    kCFCharacterSetNonBase,
    kCFCharacterSetDecomposable,
    kCFCharacterSetAlphaNumeric,
    kCFCharacterSetPunctuation,
    kCFCharacterSetCapitalizedLetter = 13,
    kCFCharacterSetSymbol = 14,
    kCFCharacterSetNewline = 15,
    kCFCharacterSetIllegal = 12
};
```

Constants

`kCFCharacterSetControl`
: Control character set (Unicode General Category Cc and Cf).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetWhitespace`
: Whitespace character set (Unicode General Category Zs and U0009 CHARACTER TABULATION).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetWhitespaceAndNewline`
: Whitespace and Newline character set (Unicode General Category Z*, `U000A ~ U000D`, and `U0085`).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetDecimalDigit`
: Decimal digit character set.

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetLetter`
: Letter character set (Unicode General Category L* & M*).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetLowercaseLetter`
: Lowercase character set (Unicode General Category Ll).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetUppercaseLetter`
: Uppercase character set (Unicode General Category Lu and Lt).

  Available in Mac OS X v10.0 and later.

  Declared in `CFCharacterSet.h`.

`kCFCharacterSetNonBase`
Non-base character set (Unicode General Category M*).

Available in Mac OS X v10.0 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetDecomposable`
Canonically decomposable character set.

Available in Mac OS X v10.0 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetAlphaNumeric`
Alpha Numeric character set (Unicode General Category L*, M*, & N*).

Available in Mac OS X v10.0 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetPunctuation`
Punctuation character set (Unicode General Category P*).

Available in Mac OS X v10.0 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetCapitalizedLetter`
Titlecase character set (Unicode General Category Lt).

Available in Mac OS X v10.2 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetSymbol`
Symbol character set (Unicode General Category S*).

Available in Mac OS X v10.3 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetNewline`
Newline character set (`U000A ~ U000D`, `U0085`, `U2028`, and `U2029`).

Available in Mac OS X v10.5 and later.

Declared in `CFCharacterSet.h`.

`kCFCharacterSetIllegal`
Illegal character set.

Available in Mac OS X v10.0 and later.

Declared in `CFCharacterSet.h`.

Discussion
Use these constants with the `CFCharacterSetGetPredefined` (page 164) function to get one of the predefined character sets.

Declared In
`CFCharacterSet.h`

CFData Reference

Derived From:	CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFData.h
Companion guides	Binary Data Programming Guide for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFData and its derived mutable type, CFMutableData, provide support for data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Core Foundation objects. CFData creates static data objects, and CFMutableData creates dynamic data objects. Data objects are typically used for raw data storage.

You use the `CFDataCreate` (page 172) and `CFDataCreateCopy` (page 173) functions to create static data objects. These functions make a new copy of the supplied data. To create a data object that uses the supplied buffer instead of making a separate copy, use the `CFDataCreateWithBytesNoCopy` (page 174) function. You use the `CFDataGetBytes` (page 176) function to retrieve the bytes and the `CFDataGetLength` (page 176) function to get the length of the bytes.

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

Functions by Task

Creating a CFData Object

`CFDataCreate` (page 172)
Creates an immutable CFData object using data copied from a specified byte buffer.

`CFDataCreateCopy` (page 173)
Creates an immutable copy of a CFData object.

`CFDataCreateWithBytesNoCopy` (page 174)
Creates an immutable CFData object from an external (client-owned) byte buffer.

Examining a CFData Object

`CFDataGetBytePtr` (page 175)
Returns a read-only pointer to the bytes of a CFData object.

`CFDataGetBytes` (page 176)
Copies the byte contents of a CFData object to an external buffer.

`CFDataGetLength` (page 176)
Returns the number of bytes contained by a CFData object.

Getting the CFData Type ID

Functions

CFDataGetTypeID

Returns the type identifier for the CFData opaque type.

```
CFTypeID CFDataGetTypeID (
   void
);
```

Return Value
The type identifier for the CFData opaque type.

Discussion
CFMutableData objects have the same type identifier as CFData objects.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFLocalServer
MoreSCF
RecentItems

Declared In
`CFData.h`

CFDataCreate

Creates an immutable CFData object using data copied from a specified byte buffer.

```
CFDataRef CFDataCreate (
   CFAllocatorRef allocator,
   const UInt8 *bytes,
   CFIndex length
);
```

Parameters

allocator

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

bytes

A pointer to the byte buffer that contains the raw data to be copied into *theData*.

length

The number of bytes in the buffer (*bytes*).

Return Value

A new CFData object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

You must supply a count of the bytes in the buffer. This function always copies the bytes in the provided buffer into internal storage.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioCDSample

BackgroundExporter

BasicInputMethod

CFHostSample

ProfileSystem

Declared In

`CFData.h`

CFDataCreateCopy

Creates an immutable copy of a CFData object.

```
CFDataRef CFDataCreateCopy (
   CFAllocatorRef allocator,
   CFDataRef theData
);
```

Parameters

allocator

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

theData

The CFData object to copy.

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

Discussion
The resulting object has the same byte contents as the original object, but it is always immutable. 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 retain count without making a true copy. To the caller, however, the resulting object is a true immutable copy, except the operation was more efficient.

Use this function when you need to pass a CFData object into another function by value (not reference).

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BackgroundExporter

Declared In
`CFData.h`

CFDataCreateWithBytesNoCopy

Creates an immutable CFData object from an external (client-owned) byte buffer.

```
CFDataRef CFDataCreateWithBytesNoCopy (
   CFAllocatorRef allocator,
   const UInt8 *bytes,
   CFIndex length,
   CFAllocatorRef bytesDeallocator
);
```

Parameters

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

bytes
A pointer to the byte buffer to be used as the backing store of the CFData object.

length
The number of bytes in the buffer *bytes*.

bytesDeallocator
The allocator to use to deallocate the external buffer when the CFData object is deallocated. If the default allocator is suitable for this purpose, pass `NULL` or `kCFAllocatorDefault`. If you do not want the created CFData object to deallocate the buffer (that is, you assume responsibility for freeing it yourself), pass `kCFAllocatorNull`.

Return Value
A new CFData object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an immutable CFData object from a buffer of unstructured bytes. 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 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 bytes.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BetterAuthorizationSample
BSDLLCTest

Declared In
`CFData.h`

CFDataGetBytePtr

Returns a read-only pointer to the bytes of a CFData object.

```
const UInt8 * CFDataGetBytePtr (
   CFDataRef theData
);
```

Parameters
theData
The CFData object to examine.

Return Value
A read-only pointer to the bytes associated with *theData*.

Discussion
This function is guaranteed to return a pointer to a CFData object's internal bytes. CFData, unlike CFString, does not hide its internal storage.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BackgroundExporter
CFHostSample
CFLocalServer
MoreSCF
String

Declared In
`CFData.h`

CFDataGetBytes

Copies the byte contents of a CFData object to an external buffer.

```
void CFDataGetBytes (
   CFDataRef theData,
   CFRange range,
   UInt8 *buffer
);
```

Parameters

theData

The CFData object to examine.

range

The range of bytes in *theData* to get. To get all of the contents, pass `CFRangeMake(0,CFDataGetLength(theData))`.

buffer

A pointer to the byte buffer of length `range.length` that is allocated on the stack or heap. On return, the buffer contains the requested range of bytes.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

BasicInputMethod

BSDLLCTest

GetPrimaryMACAddress

RecentItems

Declared In

`CFData.h`

CFDataGetLength

Returns the number of bytes contained by a CFData object.

```
CFIndex CFDataGetLength (
   CFDataRef theData
);
```

Parameters

theData

The CFData object to examine.

Return Value

An index that specifies the number of bytes in *theData*.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code
BetterAuthorizationSample
CFLocalServer
MoreSCF
Preferences
RecentItems

Declared In
`CFData.h`

Data Types

CFDataRef

A reference to an immutable CFData object.

```
typedef const struct __CFData *CFDataRef;
```

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

Declared In
`CFData.h`

CFDate Reference

Derived From:	CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFDate.h
Companion guides	Date and Time Programming Guide for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFDate objects store dates and times that can be compared to other dates and times. CFDate objects are immutable—there is no mutable counterpart for this opaque type.

CFDate provides functions for creating dates, comparing dates, and computing intervals. You use the `CFDateCreate` (page 180) function to create CFDate objects. You use the `CFDateCompare` (page 179) function to compare two dates, and the `CFDateGetTimeIntervalSinceDate` (page 181) function to compute a time interval. Additional functions for managing dates and times are described in *Time Utilities Reference*

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

Functions

CFDateCompare

Compares two CFDate objects and returns a comparison result.

```
CFComparisonResult CFDateCompare (
   CFDateRef theDate,
   CFDateRef otherDate,
   void *context
);
```

Parameters

theDate

The date to compare to *otherDate*.

otherDate
The date to compare to *theDate*.

context
Unused. Pass NULL.

Return Value
A CFComparisonResult (page 802) value that indicates whether *theDate* is equal to, less than, or greater than *otherDate*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFDate.h

CFDateCreate

Creates a CFDate object given an absolute time.

```
CFDateRef CFDateCreate (
   CFAllocatorRef allocator,
   CFAbsoluteTime at
);
```

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

at
The absolute time to convert to a CFDate object.

Return Value
A date object that represents the absolute time *at*. Ownership follows the Create Rule.

Discussion
CFDate objects must always be created using absolute time. Time intervals are not supported.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFPrefTopScores
NSOperationSample

Declared In
CFDate.h

CFDateGetAbsoluteTime

Returns a CFDate object's absolute time.

```
CFAbsoluteTime CFDateGetAbsoluteTime (
   CFDateRef theDate
);
```

Parameters

theDate

The date to examine.

Return Value

The absolute time of *theDate*.

Discussion

Absolute time is measured in seconds relative to the absolute reference date of Jan 1 2001 00:00:00 GMT. A positive value represents a date after the reference date, a negative value represents a date before it. For example, the absolute time -32940326 is equivalent to December 16th, 1999 at 17:54:34.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFDate.h

CFDateGetTimeIntervalSinceDate

Returns the number of elapsed seconds between the given CFDate objects.

```
CFTimeInterval CFDateGetTimeIntervalSinceDate (
   CFDateRef theDate,
   CFDateRef otherDate
);
```

Parameters

theDate

The date to compare to *otherDate*.

otherDate

The date to compare to *theDate*.

Return Value

The number of elapsed seconds between *theDate* and *otherDate*. The result is positive if *theDate* is later than *otherDate*.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFDate.h

CFDateGetTypeID

Returns the type identifier for the CFDate opaque type.

```
CFTypeID CFDateGetTypeID (
   void
);
```

Return Value
The type identifier for the CFDate opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFFTPSample
CFPrefTopScores

Declared In
`CFDate.h`

Data Types

CFDateRef

A reference to an immutable CFDate object.

```
typedef const struct __CFDate *CFDateRef;
```

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

Declared In
`CFDate.h`

CFDateFormatter Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFDateFormatter.h
Companion guide	Data Formatting Guide for Core Foundation

Overview

CFDateFormatter objects format the textual representations of CFDate and CFAbsoluteTime objects, and convert textual representations of dates and times into CFDate and CFAbsoluteTime objects. You can express the representation of dates and times very flexibly, for example "Thu 22 Dec 1994" is just as acceptable as "12/22/94." You specify how strings are formatted and parsed by setting a format string and other properties of a CFDateFormatter object. The format of the format string itself is defined by Unicode Technical Standard #35.

The CFDateFormatter opaque type is available in Mac OS X v10.3 and later.

Functions by Task

Creating a Date Formatter

`CFDateFormatterCreate` (page 185)
Creates a new CFDateFormatter object, localized to the given locale, which will format dates to the given date and time styles.

Configuring a Date Formatter

`CFDateFormatterSetFormat` (page 192)
Sets the format string of the given date formatter to the specified value.

`CFDateFormatterSetProperty` (page 192)
Sets a date formatter property using a key-value pair.

Parsing Strings

`CFDateFormatterCreateDateFromString` (page 187)
Returns a date object representing a given string.

`CFDateFormatterGetAbsoluteTimeFromString` (page 189)
Returns an absolute time object representing a given string.

Creating Strings From Data

`CFDateFormatterCreateStringWithAbsoluteTime` (page 188)
Returns a string representation of the given absolute time using the specified date formatter.

`CFDateFormatterCreateStringWithDate` (page 188)
Returns a string representation of the given date using the specified date formatter.

`CFDateFormatterCreateDateFormatFromTemplate` (page 186)
Returns a localized date format string representing the given date format components arranged appropriately for the specified locale.

Getting Information About a Date Formatter

`CFDateFormatterCopyProperty` (page 184)
Returns a copy of a date formatter's value for a given key.

`CFDateFormatterGetDateStyle` (page 190)
Returns the date style used to create the given date formatter object.

`CFDateFormatterGetFormat` (page 190)
Returns a format string for the given date formatter object.

`CFDateFormatterGetLocale` (page 191)
Returns the locale object used to create the given date formatter object.

`CFDateFormatterGetTimeStyle` (page 191)
Returns the time style used to create the given date formatter object.

Getting the CFDateFormatter Type ID

`CFDateFormatterGetTypeID` (page 191)
Returns the type identifier for CFDateFormatter.

Functions

CFDateFormatterCopyProperty

Returns a copy of a date formatter's value for a given key.

```
CFTypeRef CFDateFormatterCopyProperty (
   CFDateFormatterRef formatter,
   CFStringRef key
);
```

Parameters

formatter

The date formatter to examine.

key

The property key for the value to obtain. See "Date Formatter Property Keys" (page 195) for a description of possible values for this parameter.

Return Value

A CFType object that is a copy of the property value for *key*, or `NULL` if there is no value specified for *key*. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFDateFormatter.h`

CFDateFormatterCreate

Creates a new CFDateFormatter object, localized to the given locale, which will format dates to the given date and time styles.

```
CFDateFormatterRef CFDateFormatterCreate (
   CFAllocatorRef allocator,
   CFLocaleRef locale,
   CFDateFormatterStyle dateStyle,
   CFDateFormatterStyle timeStyle
);
```

Parameters

alloc

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

locale

The locale to use for localization. If `NULL` uses the default system local. Use `CFLocaleCopyCurrent` (page 242) to specify the locale of the current user.

dateStyle

The date style to use when formatting dates. See "Date Formatter Styles" (page 193) for possible values.

timeStyle

The time style to use when formatting times. See "Date Formatter Styles" (page 193) for possible values.

Return Value

A new date formatter, localized to the given locale, which will format dates to the given date and time styles. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
You can use `kCFDateFormatterNoStyle` to suppress output for the date or time. The following code fragment illustrates the creation and use of a date formatter that only outputs the date information (memory management is omitted for clarity).

```
CFLocaleRef locale = CFLocaleCreate(kCFAllocatorDefault, CFSTR("en_GB"));

CFDateFormatterRef formatter = CFDateFormatterCreate(
        kCFAllocatorDefault, locale, kCFDateFormatterMediumStyle,
kCFDateFormatterNoStyle);

CFDateRef date = CFDateCreate(kCFAllocatorDefault, 123456);
CFStringRef dateAsString = CFDateFormatterCreateStringWithDate (
        kCFAllocatorDefault, formatter, date);

CFShow(dateAsString);
// outputs "2 Jan 2001"
```

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CFFTPSample
CFPrefTopScores
FSMegaInfo

Declared In
`CFDateFormatter.h`

CFDateFormatterCreateDateFormatFromTemplate

Returns a localized date format string representing the given date format components arranged appropriately for the specified locale.

```
CFStringRef CFDateFormatterCreateDateFormatFromTemplate (
    CFAllocatorRef allocator,
    CFStringRef template,
    CFOptionFlags options,
    CFLocaleRef locale
);
```

Parameters

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

template
A string containing date format patterns (such as “MM” or “h”).

For full details, see Unicode Technical Standard #35.

options
No options are currently defined—pass `0`.

locale
The locale for which the template is required.

Return Value
A localized date format string representing the date format components given in *template*, arranged appropriately for the locale specified by *locale*. Ownership follows the Create Rule.

The returned string may not contain exactly those components given in *template*, but may—for example—have locale-specific adjustments applied.

Discussion
Different locales have different conventions for the ordering of date components. You use this method to get an appropriate format string for a given set of components for a specified locale (typically you use the current locale—see CFLocaleCopyCurrent (page 242)).

The following example shows the difference between the date formats for British and American English:

```
CFStringRef dateComponents = CFSTR("yMMMMd");

CFLocaleRef usLocale = CFLocaleCreate(NULL, CFSTR("en_US"));
CFStringRef usDateFormatString =
    CFDateFormatterCreateDateFormatFromTemplate(NULL, dateComponents, 0,
usLocale);
// Date format for English (United States): MMMM d, y

CFLocaleRef gbLocale = CFLocaleCreate(NULL, CFSTR("en_GB"));
CFStringRef gbDateFormatString =
    CFDateFormatterCreateDateFormatFromTemplate(NULL, dateComponents, 0,
gbLocale);
// Date format for English (United Kingdom): d MMMM y
```

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFDateFormatter.h

CFDateFormatterCreateDateFromString

Returns a date object representing a given string.

```
CFDateRef CFDateFormatterCreateDateFromString (
   CFAllocatorRef allocator,
   CFDateFormatterRef formatter,
   CFStringRef string,
   CFRange *rangep
);
```

Parameters

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

formatter
The date formatter object to use to parse *string*.

string
The string that contains the date.

rangep

A reference to the range within the string specifying the substring to be parsed. If `NULL`, the whole string is parsed. Upon return, contains the range that defines the extent of the parse (may be less than the given range).

Return Value
A new date that represents *string*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

CFDateFormatterCreateStringWithAbsoluteTime

Returns a string representation of the given absolute time using the specified date formatter.

```
CFStringRef CFDateFormatterCreateStringWithAbsoluteTime (
   CFAllocatorRef allocator,
   CFDateFormatterRef formatter,
   CFAbsoluteTime at
);
```

Parameters

alloc

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

formatter

The date formatter object that specifies the format of the returned string.

at

The absolute time for which to generate a string representation.

Return Value
A new string that represents *at* in the specified format. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
FSMegaInfo

Declared In
`CFDateFormatter.h`

CFDateFormatterCreateStringWithDate

Returns a string representation of the given date using the specified date formatter.

```
CFStringRef CFDateFormatterCreateStringWithDate (
   CFAllocatorRef allocator,
   CFDateFormatterRef formatter,
   CFDateRef date
);
```

Parameters

alloc

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

formatter

The date formatter object that specifies the format of the returned string.

date

The date object for which to create a string representation.

Return Value

A new string that represents *date* in the specified format. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CFFTPSample

CFPrefTopScores

Declared In

`CFDateFormatter.h`

CFDateFormatterGetAbsoluteTimeFromString

Returns an absolute time object representing a given string.

```
Boolean CFDateFormatterGetAbsoluteTimeFromString (
   CFDateFormatterRef formatter,
   CFStringRef string,
   CFRange *rangep,
   CFAbsoluteTime *atp
);
```

Parameters

formatter

The date formatter object to use to parse *string*.

string

The string that contains the time to be parsed.

rangep

Reference to the range within the string specifying the substring to be parsed. If `NULL`, the whole string is parsed. On return, the range that defines the extent of the parse (may be less than the given range).

atp

An absolute time value, returned by reference, that represents *string*. Ownership follows the Get Rule.

Return Value
`true` if the string was parsed successfully, otherwise `false`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

CFDateFormatterGetDateStyle

Returns the date style used to create the given date formatter object.

```
CFDateFormatterStyle CFDateFormatterGetDateStyle (
   CFDateFormatterRef formatter
);
```

Parameters

formatter
The date formatter to examine.

Return Value
The date style used to create *formatter*.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

CFDateFormatterGetFormat

Returns a format string for the given date formatter object.

```
CFStringRef CFDateFormatterGetFormat (
   CFDateFormatterRef formatter
);
```

Parameters

formatter
The date formatter to examine.

Return Value
The format string for *formatter* as was specified by calling the `CFDateFormatterSetFormat` (page 192) function, or derived from the date formatter's date or time styles. Ownership follows the Get Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

CFDateFormatterGetLocale

Returns the locale object used to create the given date formatter object.

```
CFLocaleRef CFDateFormatterGetLocale (
   CFDateFormatterRef formatter
);
```

Parameters

formatter
The date formatter object to examine.

Return Value
The locale object used to create *formatter*. Ownership follows the Get Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFDateFormatter.h

CFDateFormatterGetTimeStyle

Returns the time style used to create the given date formatter object.

```
CFDateFormatterStyle CFDateFormatterGetTimeStyle (
   CFDateFormatterRef formatter
);
```

Parameters

formatter
The date formatter to examine.

Return Value
The time style used to create *formatter*.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFDateFormatter.h

CFDateFormatterGetTypeID

Returns the type identifier for CFDateFormatter.

```
CFTypeID CFDateFormatterGetTypeID (
   void
);
```

Return Value
The type identifier for the CFDateFormatter opaque type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFDateFormatter.h

CFDateFormatterSetFormat

Sets the format string of the given date formatter to the specified value.

```
void CFDateFormatterSetFormat (
   CFDateFormatterRef formatter,
   CFStringRef formatString
);
```

Parameters

formatter
The date formatter to modify.

formatString
The format string for *formatter*. The syntax of this string is defined by Unicode Technical Standard #35..

Discussion
The format string may override other properties previously set using other functions. If this function is not called, the default value of the format string is derived from the date formatter's date and time styles.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFDateFormatter.h

CFDateFormatterSetProperty

Sets a date formatter property using a key-value pair.

```
void CFDateFormatterSetProperty (
   CFDateFormatterRef formatter,
   CFStringRef key,
   CFTypeRef value
);
```

Parameters

formatter
The date formatter to modify.

key
The name of the property to set. See "Date Formatter Property Keys" (page 195) for a description of possible values for this parameter.

value
The value for *key*. This should be a CFType object corresponding to the specified key.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFDateFormatter.h

Data Types

CFDateFormatterRef

A reference to a CFDateFormatter object.

```
typedef struct __CFDateFormatter *CFDateFormatterRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

CFDateFormatterStyle

Data type for predefined date and time format styles.

```
typedef CFIndex CFDateFormatterStyle;
```

Discussion
For possible values, see "Date Formatter Styles" (page 193).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFDateFormatter.h`

Constants

Date Formatter Styles

Predefined date and time format styles.

```
enum {
    kCFDateFormatterNoStyle = 0,
    kCFDateFormatterShortStyle = 1,
    kCFDateFormatterMediumStyle = 2,
    kCFDateFormatterLongStyle = 3,
    kCFDateFormatterFullStyle = 4
};
```

Constants

`kCFDateFormatterNoStyle`

Specifies no output.

You use this constant to suppress output for the date or time (see `CFDateFormatterCreate` (page 185) for more details).

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortStyle`

Specifies a short style, typically numeric only, such as “11/23/37” or “3:30pm”.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterMediumStyle`

Specifies a medium style, typically with abbreviated text, such as “Nov 23, 1937”.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterLongStyle`

Specifies a long style, typically with full text, such as “November 23, 1937” or “3:30:32pm”.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterFullStyle`

Specifies a full style with complete details, such as “Tuesday, April 12, 1952 AD” or “3:30:42pm PST”.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

Discussion

The format for these date and time styles is not exact because they depend on the locale, user preference settings, and the operating system version. Do not use these constants if you want an exact format, for example if you are parsing an external data file which contains date information in a fixed format. There are several different “lengths” of the formats:

- "long" era names, for example "Anno Domini" instead of "AD"
- "very short" names for months and weekdays; for example, "F" instead of "Friday"
- "standalone" names for months and weekdays (for some locales or languages, a month name displayed in isolation needs to be written differently than a month name within a displayed date)
- names of quarters; for example, "Q2" for a short quarter name

Declared In

`CFDateFormatter.h`

Date Formatter Property Keys

Keys used in key-value pairs to discover and specify the value of date formatter properties—used in conjunction with `CFDateFormatterCopyProperty` (page 184) and `CFDateFormatterSetProperty` (page 192).

```
const CFStringRef kCFDateFormatterIsLenient;
const CFStringRef kCFDateFormatterTimeZone;
const CFStringRef kCFDateFormatterCalendarName;
const CFStringRef kCFDateFormatterDefaultFormat;

const CFStringRef kCFDateFormatterTwoDigitStartDate;
const CFStringRef kCFDateFormatterDefaultDate;
const CFStringRef kCFDateFormatterCalendar;
const CFStringRef kCFDateFormatterEraSymbols;
const CFStringRef kCFDateFormatterMonthSymbols;
const CFStringRef kCFDateFormatterShortMonthSymbols;
const CFStringRef kCFDateFormatterWeekdaySymbols;
const CFStringRef kCFDateFormatterShortWeekdaySymbols;
const CFStringRef kCFDateFormatterAMSymbol;
const CFStringRef kCFDateFormatterPMSymbol;

const CFStringRef kCFDateFormatterLongEraSymbols;
const CFStringRef kCFDateFormatterVeryShortMonthSymbols;
const CFStringRef kCFDateFormatterStandaloneMonthSymbols;
const CFStringRef kCFDateFormatterShortStandaloneMonthSymbols;
const CFStringRef kCFDateFormatterVeryShortStandaloneMonthSymbols;
const CFStringRef kCFDateFormatterVeryShortWeekdaySymbols;
const CFStringRef kCFDateFormatterStandaloneWeekdaySymbols;
const CFStringRef kCFDateFormatterShortStandaloneWeekdaySymbols;
const CFStringRef kCFDateFormatterVeryShortStandaloneWeekdaySymbols;
const CFStringRef kCFDateFormatterQuarterSymbols;
const CFStringRef kCFDateFormatterShortQuarterSymbols;
const CFStringRef kCFDateFormatterStandaloneQuarterSymbols;
const CFStringRef kCFDateFormatterShortStandaloneQuarterSymbols;
const CFStringRef kCFDateFormatterGregorianStartDate;
```

Constants

`kCFDateFormatterIsLenient`

Specifies the lenient property, a CFBoolean object where a true value indicates that the parsing of strings into date or absolute time values will be fuzzy.

The formatter will use heuristics to guess at the date which is intended by the string. As with any guessing, it may get the result date wrong (that is, a date other than that which was intended).

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterTimeZone`

Specifies the time zone property, a CFTimeZone object.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterCalendarName`
Specifies the calendar name, a CFString object.

With Mac OS X version 10.3, `kCFGregorianCalendar` (page 198) is the only possible value. With Mac OS X version 10.4, `kCFGregorianCalendar` and other calendar names are specified by CFLocale.

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterDefaultFormat`
The original format string for the formatter (given the date & time style and locale specified at creation).

Available in Mac OS X v10.3 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterTwoDigitStartDate`
Specifies the property representing the date from which two-digit years start, a CFDate object.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterDefaultDate`
Specifies the default date property, a CFDate object.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterCalendar`
Specifies the calendar property, a CFCalendar object.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterEraSymbols`
Specifies the era symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterMonthSymbols`
Specifies the month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortMonthSymbols`
Specifies the short month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterWeekdaySymbols`
Specifies the weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortWeekdaySymbols`
Specifies the short weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterAMSymbol`
Specifies the AM symbol property, a CFString object.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterPMSymbol`
Specifies the PM symbol property, a CFString object.

Available in Mac OS X v10.4 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterLongEraSymbols`
Specifies the long era symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterVeryShortMonthSymbols`
Specifies the very short month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterStandaloneMonthSymbols`
Specifies the standalone month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortStandaloneMonthSymbols`
Specifies the short standalone month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterVeryShortStandaloneMonthSymbols`
Specifies the very short standalone month symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterVeryShortWeekdaySymbols`
Specifies the very short weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterStandaloneWeekdaySymbols`
Specifies the standalone weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortStandaloneWeekdaySymbols`
Specifies the short standalone weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterVeryShortStandaloneWeekdaySymbols`
Specifies the very short standalone weekday symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterQuarterSymbols`
Specifies the quarter symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortQuarterSymbols`
Specifies the short quarter symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterStandaloneQuarterSymbols`
Specifies the standalone quarter symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterShortStandaloneQuarterSymbols`
Specifies the short standalone quarter symbols property, a CFArray of CFString objects.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

`kCFDateFormatterGregorianStartDate`
Specifies the Gregorian start date property, a CFDate object.

This is used to specify the start date for the Gregorian calendar switch from the Julian calendar. Different locales switched at different times. Normally you should just accept the locale's default date for the switch.

Available in Mac OS X v10.5 and later.

Declared in `CFDateFormatter.h`.

Discussion
The values for these keys are all CFType objects. The specific types for each key are specified above.

Declared In
`CFDateFormatter.h`

Calendar Names

Calendar names used by CFDateFormatter.

```
const CFStringRef kCFGregorianCalendar;
```

Constants
`kCFGregorianCalendar`
The name of the calendar currently supported by the `kCFDateFormatterCalendarName` (page 196) property.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

Declared In
`CFDateFormatter.h`

CFDictionary Reference

Derived From:	*CFPropertyList Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFDictionary.h
Companion guides	Collections Programming Topics for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFDictionary and its derived mutable type, *CFMutableDictionary Reference*, manage associations of key-value pairs. CFDictionary creates static dictionaries where you set the key-value pairs when first creating a dictionary and cannot modify them afterward; CFMutableDictionary creates dynamic dictionaries where you can add or delete key-value pairs at any time, and the dictionary automatically allocates memory as needed.

A key-value pair within a dictionary is called an entry. Each entry consists of one object that represents the key and a second object that is that key's value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by the equal callback). Internally, a dictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key.

Keys for a CFDictionary may be of any C type, however note that if you want to convert a CFPropertyList to XML, any dictionary's keys must be CFString objects.

You create static dictionaries using either the `CFDictionaryCreate` (page 205) or `CFDictionaryCreateCopy` (page 206) function. Key-value pairs are passed as parameters to `CFDictionaryCreate` (page 205). When adding key-value pairs to a dictionary, the keys and values are not copied—they are retained so they are not invalidated before the dictionary is deallocated.

CFDictionary provides functions for querying the values of a dictionary. The function `CFDictionaryGetCount` (page 207) returns the number of key-value pairs in a dictionary; the `CFDictionaryContainsValue` (page 204) function checks if a value is in a dictionary; and `CFDictionaryGetKeysAndValues` (page 208) returns a C array containing all the values and a C array containing all the keys in a dictionary.

The `CFDictionaryApplyFunction` (page 203) function lets you apply a function to all key-value pairs in a dictionary.

CFDictionary is "toll-free bridged" with its Cocoa Foundation counterpart, `NSDictionary`. 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 `NSDictionary *` parameter, you can pass in a `CFDictionaryRef`, and in a function where you see a `CFDictionaryRef` parameter, you can pass in an NSDictionary instance. This also applies to concrete subclasses of NSDictionary. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a dictionary

`CFDictionaryCreate` (page 205)
Creates an immutable dictionary containing the specified key-value pairs.

`CFDictionaryCreateCopy` (page 206)
Creates and returns a new immutable dictionary with the key-value pairs of another dictionary.

Examining a dictionary

`CFDictionaryContainsKey` (page 203)
Returns a Boolean value that indicates whether a given key is in a dictionary.

`CFDictionaryContainsValue` (page 204)
Returns a Boolean value that indicates whether a given value is in a dictionary.

`CFDictionaryGetCount` (page 207)
Returns the number of key-value pairs in a dictionary.

`CFDictionaryGetCountOfKey` (page 207)
Returns the number of times a key occurs in a dictionary.

`CFDictionaryGetCountOfValue` (page 208)
Counts the number of times a given value occurs in the dictionary.

`CFDictionaryGetKeysAndValues` (page 208)
Fills two buffers with the keys and values from a dictionary.

`CFDictionaryGetValue` (page 210)
Returns the value associated with a given key.

`CFDictionaryGetValueIfPresent` (page 210)
Returns a Boolean value that indicates whether a given value for a given key is in a dictionary, and returns that value indirectly if it exists.

Applying a function to a dictionary

`CFDictionaryApplyFunction` (page 203)
Calls a function once for each key-value pair in a dictionary.

Getting the CFDictionary type ID

`CFDictionaryGetTypeID` (page 209)
Returns the type identifier for the CFDictionary opaque type.

Functions

CFDictionaryApplyFunction

Calls a function once for each key-value pair in a dictionary.

```
void CFDictionaryApplyFunction (
   CFDictionaryRef theDict,
   CFDictionaryApplierFunction applier,
   void *context
);
```

Parameters

theDict

The dictionary to operate upon.

applier

The callback function to call once for each key-value pair in *theDict*. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. If there are keys or values which the *applier* function does not expect or cannot properly apply to, the behavior is undefined.

context

A pointer-sized program-defined value, which is passed as the third parameter to the applier function, but is otherwise unused by this function. The value must be appropriate for the *applier* function.

Discussion

If this function iterates over a mutable collection, it is unsafe for the *applier* function to change the contents of the collection.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

ColorSyncDevices

IOPrintSuperClasses

MoreSCF

Declared In

CFDictionary.h

CFDictionaryContainsKey

Returns a Boolean value that indicates whether a given key is in a dictionary.

```
Boolean CFDictionaryContainsKey (
   CFDictionaryRef theDict,
   const void *key
);
```

Parameters

theDict

The dictionary to examine.

key

The key for which to find matches in *theDict*. The key hash and equal callbacks provided when the dictionary was created, are used to compare. If the hash callback is NULL, *key* is treated as a pointer and converted to an integer. If the equal callback is NULL, pointer equality (in C, ==) is used. If *key*, or any of the keys in the dictionary, is not understood by the equal callback, the behavior is undefined.

Return Value
true if *key* is in the dictionary, otherwise false.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Utilities
MoreSCF
SeeMyFriends

Declared In
CFDictionary.h

CFDictionaryContainsValue

Returns a Boolean value that indicates whether a given value is in a dictionary.

```
Boolean CFDictionaryContainsValue (
   CFDictionaryRef theDict,
   const void *value
);
```

Parameters

theDict

The dictionary to examine.

value

The value for which to find matches in *theDict*. The value equal callback provided when the dictionary was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in the dictionary, is not understood by the equal callback, the behavior is undefined.

Return Value
true if *value* is in the dictionary, otherwise false.

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

Related Sample Code
ProfileSystem

Declared In
CFDictionary.h

CFDictionaryCreate

Creates an immutable dictionary containing the specified key-value pairs.

```
CFDictionaryRef CFDictionaryCreate (
   CFAllocatorRef allocator,
   const void **keys,
   const void **values,
   CFIndex numValues,
   const CFDictionaryKeyCallBacks *keyCallBacks,
   const CFDictionaryValueCallBacks *valueCallBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new dictionary. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

keys

A C array of the pointer-sized keys to be in the new dictionary. This value may be `NULL` if the *numValues* parameter is `0`. This C array is not changed or freed by this function. The value must be a valid pointer to a C array of at least *numValues* pointers.

values

A C array of the pointer-sized values to be in the new dictionary. This value may be `NULL` if the *numValues* parameter is `0`. This C array is not changed or freed by this function. The value must be a valid pointer to a C array of at least *numValues* elements.

numValues

The number of key-value pairs to copy from the *keys* and *values* C arrays into the new dictionary. This number will be the count of the dictionary; it must be non-negative and less than or equal to the actual number of keys or values.

keyCallBacks

A pointer to a `CFDictionaryKeyCallBacks` (page 215) structure initialized with the callbacks to use to retain, release, describe, and compare keys in the dictionary. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be `NULL`, which is treated as if a valid structure of version `0` with all fields `NULL` had been passed in. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a `CFDictionaryKeyCallBacks` (page 215) structure, the behavior is undefined. If any of the keys put into the collection is not one understood by one of the callback functions the behavior when that callback function is used is undefined.

If the collection will contain CFType objects only, then pass a pointer to `kCFTypeDictionaryKeyCallBacks` (page 218) as this parameter to use the default callback functions.

valueCallBacks

A pointer to a `CFDictionaryValueCallBacks` (page 216) structure initialized with the callbacks to use to retain, release, describe, and compare values in the dictionary. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be `NULL`, which is treated as if a valid structure of version `0` with all fields `NULL` had been passed in. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a `CFDictionaryValueCallBacks` structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions the behavior when that callback function is used is undefined.

If the collection will contain CFType objects only, then pass a pointer to `kCFTypeDictionaryValueCallBacks` (page 218) as this parameter to use the default callback functions.

Return Value
A new dictionary, 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
BetterAuthorizationSample
BSDLLCTest
MoreSCF
QTMetaData
Quartz2DBasics

Declared In
`CFDictionary.h`

CFDictionaryCreateCopy

Creates and returns a new immutable dictionary with the key-value pairs of another dictionary.

```
CFDictionaryRef CFDictionaryCreateCopy (
   CFAllocatorRef allocator,
   CFDictionaryRef theDict
);
```

Parameters

allocator

The allocator to use to allocate memory for the new dictionary. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

theDict

The dictionary to copy. The keys and values from the dictionary are copied as pointers into the new dictionary. However, the keys and values are also retained by the new dictionary. The count of the new dictionary is the same as the count of *theDict*. The new dictionary uses the same callbacks as *theDict*.

Return Value
A new dictionary that contains the same key-value pairs as *theDict*, 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.

Declared In
`CFDictionary.h`

CFDictionaryGetCount

Returns the number of key-value pairs in a dictionary.

```
CFIndex CFDictionaryGetCount (
   CFDictionaryRef theDict
);
```

Parameters

theDict

The dictionary to examine.

Return Value
The number of number of key-value pairs in *theDict*.

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

Related Sample Code
BSDLLCTest
DockBrowser
FSMegaInfo
MoreSCF
SeeMyFriends

Declared In
`CFDictionary.h`

CFDictionaryGetCountOfKey

Returns the number of times a key occurs in a dictionary.

```
CFIndex CFDictionaryGetCountOfKey (
   CFDictionaryRef theDict,
   const void *key
);
```

Parameters

theDict

The dictionary to examine.

key

The key for which to find matches in *theDict*. The key hash and equal callbacks provided when the dictionary was created are used to compare. If the hash callback was `NULL`, the key is treated as a pointer and converted to an integer. If the equal callback was `NULL`, pointer equality (in C, ==) is used. If *key*, or any of the keys in the dictionary, is not understood by the equal callback, the behavior is undefined.

Return Value
Returns `1` if a matching key is used by the dictionary, otherwise `0`.

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

Declared In
`CFDictionary.h`

CFDictionaryGetCountOfValue

Counts the number of times a given value occurs in the dictionary.

```
CFIndex CFDictionaryGetCountOfValue (
   CFDictionaryRef theDict,
   const void *value
);
```

Parameters

`theDict`
The dictionary to examine.

`value`
The value for which to find matches in *`theDict`*. The value equal callback provided when the dictionary was created is used to compare. If the equal callback was `NULL`, pointer equality (in C, ==) is used. If *`value`*, or any other value in the dictionary, is not understood by the equal callback, the behavior is undefined.

Return Value
The number of times the *`value`* occurs in *`theDict`*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFDictionary.h`

CFDictionaryGetKeysAndValues

Fills two buffers with the keys and values from a dictionary.

```
void CFDictionaryGetKeysAndValues (
   CFDictionaryRef theDict,
   const void **keys,
   const void **values
);
```

Parameters

`theDict`
The dictionary to examine.

keys

A C array of pointer-sized values that, on return, is filled with keys from the *theDict*. The keys and values C arrays are parallel to each other (that is, the items at the same indices form a key-value pair from the dictionary). This value must be a valid pointer to a C array of the appropriate type and size (that is, a size equal to the count of *theDict*), or NULL if the keys are not required. If the keys are Core Foundation objects, ownership follows the Get Rule.

values

A C array of pointer-sized values that, on return, is filled with values from the *theDict*. The keys and values C arrays are parallel to each other (that is, the items at the same indices form a key-value pair from the dictionary). This value must be a valid pointer to a C array of the appropriate type and size (that is, a size equal to the count of *theDict*), or NULL if the values are not required. If the values are Core Foundation objects, ownership follows the Get Rule.

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

Related Sample Code
BSDLLCTest
DockBrowser
FSMegaInfo
MoreSCF
SeeMyFriends

Declared In
CFDictionary.h

CFDictionaryGetTypeID

Returns the type identifier for the CFDictionary opaque type.

```
CFTypeID CFDictionaryGetTypeID (
   void
);
```

Return Value
The type identifier for the CFDictionary opaque type.

Discussion
CFMutableDictionary objects have the same type identifier as CFDictionary objects.

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

Related Sample Code
BSDLLCTest
HID Calibrator
HID Config Save
HID Utilities
MoreSCF

Declared In
CFDictionary.h

CFDictionaryGetValue

Returns the value associated with a given key.

```
const void * CFDictionaryGetValue (
   CFDictionaryRef theDict,
   const void *key
);
```

Parameters

theDict

The dictionary examine.

key

The key for which to find a match in *theDict*. The key hash and equal callbacks provided when the dictionary was created are used to compare. If the hash callback was NULL, the key is treated as a pointer and converted to an integer. If the equal callback was NULL, pointer equality (in C, ==) is used. If *key*, or any of the keys in *theDict*, is not understood by the equal callback, the behavior is undefined.

Return Value

The value associated with *key* in *theDict*, or NULL if no key-value pair matching *key* exists. Since NULL is also a valid value in some dictionaries, use CFDictionaryGetValueIfPresent (page 210) to distinguish between a value that is not found, and a NULL value. If the value is a Core Foundation object, ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

audioburntest

bulkerase

databurntest

GLUT

MoreSCF

Declared In

CFDictionary.h

CFDictionaryGetValueIfPresent

Returns a Boolean value that indicates whether a given value for a given key is in a dictionary, and returns that value indirectly if it exists.

```
Boolean CFDictionaryGetValueIfPresent (
   CFDictionaryRef theDict,
   const void *key,
   const void **value
);
```

Parameters

theDict

The dictionary to examine.

key

The key for which to find a match in *theDict*. The key hash and equal callbacks provided when the dictionary was created are used to compare. If the hash callback was NULL, key is treated as a pointer and converted to an integer. If the equal callback was NULL, pointer equality (in C, ==) is used. If key, or any of the keys in *theDict*, is not understood by the equal callback, the behavior is undefined.

value

A pointer to memory which, on return, is filled with the pointer-sized value if a matching key is found. If no key match is found, the contents of the storage pointed to by this parameter are undefined. This value may be NULL, in which case the value from the dictionary is not returned (but the return value of this function still indicates whether or not the key-value pair was present). If the value is a Core Foundation object, ownership follows the Get Rule.

Return Value
true if a matching key was found, otherwise false.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Dumper
HID Utilities
HID Utilities Source

Declared In
CFDictionary.h

Callbacks

CFDictionaryApplierFunction

Prototype of a callback function that may be applied to every key-value pair in a dictionary.

```
typedef void (*CFDictionaryApplierFunction) (
   const void *key,
   const void *value,
   void *context
);
```

If you name your function MyCallBack, you would declare it like this:

```
void MyCallBack (
   const void *key,
   const void *value,
   void *context
);
```

Parameters

key

The key associated with the current key-value pair.

value

The value associated with the current key-value pair.

context

The program-defined context parameter given to the apply function.

Discussion

This callback is passed to the `CFDictionaryApplyFunction` (page 203) function which iterates over the key-value pairs in a dictionary and applies the behavior defined in the applier function to each key-value pair in a dictionary.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFDictionary.h`

CFDictionaryCopyDescriptionCallBack

Prototype of a callback function used to get a description of a value or key in a dictionary.

```
typedef CFStringRef (*CFDictionaryCopyDescriptionCallBack)(
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallback (
   const void *value
);
```

Parameters

value

The value to be described.

Return Value

A text description of *value*.

Discussion

This callback is passed to `CFDictionaryCreate` (page 205) in a `CFDictionaryKeyCallBacks` (page 215) structure or `CFDictionaryValueCallBacks` (page 216). This callback is used by the `CFCopyDescription` (page 652) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFDictionary.h`

CFDictionaryEqualCallBack

Prototype of a callback function used to determine if two values or keys in a dictionary are equal.

```
typedef Boolean  (*CFDictionaryEqualCallBack) (
   const void *value1,
   const void *value2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
   const void *value1,
   const void *value2
);
```

Parameters

value1

A value in the dictionary.

value2

Another value in the dictionary.

Discussion

This callback is passed to `CFDictionaryCreate` (page 205) in a `CFDictionaryKeyCallBacks` (page 215) and `CFDictionaryValueCallBacks` (page 216) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFDictionary.h`

CFDictionaryHashCallBack

Prototype of a callback function invoked to compute a hash code for a key. Hash codes are used when key-value pairs are accessed, added, or removed from a collection.

```
typedef CFHashCode    (*CFDictionaryHashCallBack) (
   const void *value
);
```

If you name your function `MyDictionaryHashCallBack`, you would declare it like this:

```
CFHashCode MyDictionaryHashCallBack (
   const void *value
);
```

Parameters

value

The value used to compute the hash code.

Return Value

An integer that can be used as a table address in a hash table structure.

Discussion
This callback is passed to `CFDictionaryCreate` (page 205) in a `CFDictionaryKeyCallBacks` (page 215) structure.

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

Declared In
`CFDictionary.h`

CFDictionaryReleaseCallBack

Prototype of a callback function used to release a key-value pair before it's removed from a dictionary.

```
typedef void (*CFDictionaryReleaseCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator
The dictionary's allocator.

value
The value being removed from the dictionary.

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

Declared In
`CFDictionary.h`

CFDictionaryRetainCallBack

Prototype of a callback function used to retain a value or key being added to a dictionary.

```
typedef const void *(*CFDictionaryRetainCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
   CFAllocatorRef allocator,
   const void *value
```

```
);
```

Parameters

allocator

The dictionary's allocator.

value

The value being added to the dictionary.

Return Value

The value or key to store in the dictionary, which is usually the *value* parameter passed to this callback, but may be a different value if a different value should be stored in the collection.

Discussion

This callback is passed to `CFDictionaryCreate` (page 205) in a `CFDictionaryKeyCallBacks` (page 215) and `CFDictionaryValueCallBacks` (page 216) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFDictionary.h`

Data Types

CFDictionaryKeyCallBacks

This structure contains the callbacks used to retain, release, describe, and compare the keys in a dictionary.

```
struct CFDictionaryKeyCallBacks {
    CFIndex version;
    CFDictionaryRetainCallBack retain;
    CFDictionaryReleaseCallBack release;
    CFDictionaryCopyDescriptionCallBack copyDescription;
    CFDictionaryEqualCallBack equal;
    CFDictionaryHashCallBack hash;
};
typedef struct CFDictionaryKeyCallBacks CFDictionaryKeyCallBacks;
```

Fields

`version`

The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is 0.

`retain`

The callback used to retain each key as they are added to the collection. This callback returns the value to use as the key in the dictionary, which is usually the value parameter passed to this callback, but may be a different value if a different value should be used as the key. If `NULL`, keys are not retained. See `CFDictionaryRetainCallBack` (page 214) for a descriptions of this function's parameters.

`release`
: The callback used to release keys as they are removed from the dictionary. If `NULL`, keys are not released. See `CFDictionaryReleaseCallBack` (page 214) for a description of this callback.

`copyDescription`
: The callback used to create a descriptive string representation of each key in the dictionary. If `NULL`, the collection will create a simple description of each key. See `CFDictionaryCopyDescriptionCallBack` (page 212) for a description of this callback.

`equal`
: The callback used to compare keys in the dictionary for equality. If `NULL`, the collection will use pointer equality to compare keys in the collection. See `CFDictionaryEqualCallBack` (page 213) for a description of this callback.

`hash`
: The callback used to compute a hash code for keys as they are used to access, add, or remove values in the dictionary. If `NULL`, the collection computes a hash code by converting the pointer value to an integer. See `CFDictionaryHashCallBack` (page 213) for a description of this callback.

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

Declared In
`CFDictionary.h`

CFDictionaryRef

A reference to an immutable dictionary object.

```
typedef const struct __CFDictionary *CFDictionaryRef;
```

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

Declared In
`CFDictionary.h`

CFDictionaryValueCallBacks

This structure contains the callbacks used to retain, release, describe, and compare the values in a dictionary.

```
struct CFDictionaryValueCallBacks {
    CFIndex version;
    CFDictionaryRetainCallBack retain;
    CFDictionaryReleaseCallBack release;
    CFDictionaryCopyDescriptionCallBack copyDescription;
    CFDictionaryEqualCallBack equal;
};
typedef struct CFDictionaryValueCallBacks CFDictionaryValueCallBacks;
```

Fields
`version`
: The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is 0.

`retain`
: The callback used to retain each value as they are added to the collection. This callback returns the value to use as the value in the dictionary, which is usually the value parameter passed to this callback, but may be a different value if a different value should be used as the value. If `NULL`, values are not retained. See `CFDictionaryRetainCallBack` (page 214) for a descriptions of this function's parameters.

`release`
: The callback used to release values as they are removed from the dictionary. If `NULL`, values are not released. See `CFDictionaryReleaseCallBack` (page 214) for a description of this callback.

`copyDescription`
: The callback used to create a descriptive string representation of each value in the dictionary. If `NULL`, the collection will create a simple description of each value. See `CFDictionaryCopyDescriptionCallBack` (page 212) for a description of this callback.

`equal`
: The callback used to compare values in the dictionary for equality. If `NULL`, the collection will use pointer equality to compare values in the collection. See `CFDictionaryEqualCallBack` (page 213) for a description of this callback.

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

Declared In
`CFDictionary.h`

Constants

Predefined Callback Structures

CFDictionary provides some predefined callbacks for your convenience.

```
const CFDictionaryKeyCallBacks kCFCopyStringDictionaryKeyCallBacks;
const CFDictionaryKeyCallBacks kCFTypeDictionaryKeyCallBacks;
const CFDictionaryValueCallBacks kCFTypeDictionaryValueCallBacks;
```

Constants

`kCFCopyStringDictionaryKeyCallBacks`

Predefined `CFDictionaryKeyCallBacks` (page 215) structure containing a set of callbacks appropriate for use when the keys of a CFDictionary are all CFString objects, which may be mutable and need to be copied in order to serve as constant keys for the values in the dictionary.

You typically use a pointer to this constant when creating a new dictionary.

Important: For performance reasons, the default `kCFCopyStringDictionaryKeyCallBacks` behavior uses `CFEqual` (page 653) which does not normalize the strings. This means that, for example, it does not consider CFStrings to be equal when they are the same but one is in pre-composed form (say, originating from a UTF-16 text file) and the other in decomposed form (say, originating from a file name). In cases where you use strings from different sources, you may want to pre-normalize the keys or else use a different set of functions to perform the comparison.

Available in Mac OS X v10.0 and later.

Declared in `CFDictionary.h`.

`kCFTypeDictionaryKeyCallBacks`

Predefined `CFDictionaryKeyCallBacks` (page 215) structure containing a set of callbacks appropriate for use when the keys of a CFDictionary are all CFType-derived objects.

The retain callback is `CFRetain`, the release callback is `CFRelease`, the copy callback is `CFCopyDescription`, the equal callback is `CFEqual`. Therefore, if you use a pointer to this constant when creating the dictionary, keys are automatically retained when added to the collection, and released when removed from the collection.

Available in Mac OS X v10.0 and later.

Declared in `CFDictionary.h`.

`kCFTypeDictionaryValueCallBacks`

Predefined `CFDictionaryValueCallBacks` (page 216) structure containing a set of callbacks appropriate for use when the values in a CFDictionary are all CFType-derived objects.

The retain callback is `CFRetain`, the release callback is `CFRelease`, the copy callback is `CFCopyDescription`, and the equal callback is `CFEqual`. Therefore, if you use a pointer to this constant when creating the dictionary, values are automatically retained when added to the collection, and released when removed from the collection.

Available in Mac OS X v10.0 and later.

Declared in `CFDictionary.h`.

CFError Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFError.h
Companion guide	Error Handling Programming Guide

Overview

A CFError object encapsulates rich and extensible error information than is possible using only an error code or error string. The core attributes of a CFError object are an error domain (represented by a string), a domain-specific error code and a user info dictionary containing application specific information. Errors are required to have a domain and an error code within that domain. The optional "userInfo" dictionary may provide additional information that might be useful for the interpretation and reporting of the error. This dictionary can even contain an “underlying” error, which is wrapped as an error bubbles up through various layers.

Several well-known domains are defined corresponding to Mach, POSIX, and OSStatus errors. In addition, CFError allows you to attach an arbitrary user info dictionary to an error object, and provides the means to return a human-readable description for the error.

In general, a method should signal an error condition by—for example—returning `false` or `NULL` rather than by the simple presence of an error object. The method can then optionally return an CFError object by reference, in order to further describe the error.

CFError is toll-free bridged to `NSError` in the Foundation framework—for more details on toll-free bridging, see Interchangeable Data Types. `NSError` has some additional guidelines which makes it easy to automatically report errors to users and even try to recover from them. See *Error Handling Programming Guide* for more information on `NSError` programming guidelines.

Functions by Task

Creating a CFError

`CFErrorCreate` (page 223)
Creates a new CFError object.

`CFErrorCreateWithUserInfoKeysAndValues` (page 224)
Creates a new CFError object using given keys and values to create the user info dictionary.

Getting Information About an Error

`CFErrorGetDomain` (page 225)
Returns the error domain for a given CFError.

`CFErrorGetCode` (page 224)
Returns the error code for a given CFError.

`CFErrorCopyUserInfo` (page 222)
Returns the user info dictionary for a given CFError.

`CFErrorCopyDescription` (page 220)
Returns a human-presentable description for a given error.

`CFErrorCopyFailureReason` (page 221)
Returns a human-presentable failure reason for a given error.

`CFErrorCopyRecoverySuggestion` (page 222)
Returns a human presentable recovery suggestion for a given error.

Getting the CFError Type ID

`CFErrorGetTypeID` (page 225)
Returns the type identifier for the CFError opaque type.

Functions

CFErrorCopyDescription

Returns a human-presentable description for a given error.

```
CFStringRef CFErrorCopyDescription (
   CFErrorRef err
);
```

Parameters

err
The CFError to examine. If this is not a valid CFError, the behavior is undefined.

Return Value
A localized, human-presentable description of *err*. This function never returns `NULL`. Ownership follows the Create Rule.

Discussion
This is a complete sentence or two which says what failed and why it failed. The structure of the description depends on the details provided in the user info dictionay. The rules for computing the return value are as follows:

1. If the value in the user info dictionary for `kCFErrorLocalizedDescriptionKey` (page 227) is not `NULL`, returns that value as-is.
2. If the value in the user info dictionary for `kCFErrorLocalizedFailureReasonKey` (page 227) is not `NULL`, generate an error from that.

The description is something like: "Operation could not be completed. " + `kCFErrorLocalizedFailureReasonKey`

3. Generate as good a user-presentable string as possible from `kCFErrorDescriptionKey` (page 227), the domain, and code.

   The description is something like like: "Operation could not be completed. Error domain/code occurred. " or "Operation could not be completed. " + `kCFErrorDescriptionKey` + " (Error domain/code)"

Toll-free bridged instances of `NSError` might provide additional behaviors for manufacturing a description string.

You should not depend on the exact contents or format of the returned string, as it might change in different releases of the operating system.

When you create a CFError, you should try to make sure the return value is human-presentable and localized by providing a value for `kCFErrorLocalizedDescriptionKey` (page 227) in the user info dictionary.

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

Related Sample Code
From A View to A Movie

Declared In
`CFError.h`

CFErrorCopyFailureReason

Returns a human-presentable failure reason for a given error.

```
CFStringRef CFErrorCopyFailureReason (
   CFErrorRef err
);
```

Parameters

err

The CFError to examine. If this is not a valid CFError, the behavior is undefined.

Return Value
A localized, human-presentable failure reason for *err*, or `NULL` if no user-presentable string is available. Ownership follows the Create Rule.

Discussion
The failure reason is a complete sentence which describes why the operation failed. In many cases this will be just the "because" part of the description (but as a complete sentence, which makes localization easier). For example, an error description "Could not save file 'Letter' in folder 'Documents' because the volume 'MyDisk' doesn't have enough space." might have a corresponding failure reason, "The volume 'MyDisk' doesn't have enough space."

By default, this function looks for a value for the `kCFErrorLocalizedFailureReasonKey` (page 227) key in the user info dictionary. Toll-free bridged instances of `NSError` might provide additional behaviors for manufacturing this value.

When you create a CFError, you should try to make sure the return value is human-presentable and localized by providing a value for `kCFErrorLocalizedFailureReasonKey` (page 227) in the user info dictionary.

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

Declared In
`CFError.h`

CFErrorCopyRecoverySuggestion

Returns a human presentable recovery suggestion for a given error.

```
CFStringRef CFErrorCopyRecoverySuggestion (
   CFErrorRef err
);
```

Parameters

err

The CFError to examine. If this is not a valid CFError, the behavior is undefined.

Return Value
A localized, human-presentable recovery suggestion for *err*, or `NULL` if no user-presentable string is available. Ownership follows the Create Rule.

Discussion
This is the string that can be displayed as the "informative" (or "secondary") message on an alert panel. For example, an error description "Could not save file 'Letter' in folder 'Documents' because the volume 'MyDisk' doesn't have enough space." might have a corresponding recovery suggestion, "Remove some files from the volume and try again."

By default, this function looks for a value for the `kCFErrorLocalizedRecoverySuggestionKey` (page 227) key in the user info dictionary. Toll-free bridged instances of `NSError` might provide additional behaviors for manufacturing this value.

When you create a CFError, you should try to make sure the return value is human-presentable and localized by providing a value for `kCFErrorLocalizedRecoverySuggestionKey` (page 227) in the user info dictionary.

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

Declared In
`CFError.h`

CFErrorCopyUserInfo

Returns the user info dictionary for a given CFError.

```
CFDictionaryRef CFErrorCopyUserInfo (
   CFErrorRef err
);
```

Parameters

err

The error to examine. If this is not a valid CFError, the behavior is undefined.

Return Value

A dictionary containing the same keys and values as in the userInfo dictionary *err* was created with. Returns an empty dictionary if `NULL` was supplied to the create function. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFError.h`

CFErrorCreate

Creates a new CFError object.

```
CFErrorRef CFErrorCreate (
   CFAllocatorRef allocator,
   CFStringRef domain,
   CFIndex code,
   CFDictionaryRef userInfo
);
```

Parameters

allocator

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

domain

A CFString that identifies the error domain. If this reference is `NULL` or is otherwise not a valid CFString, the behavior is undefined.

code

A CFIndex that identifies the error code. The code is interpreted within the context of the error domain.

userInfo

A CFDictionary created with `kCFCopyStringDictionaryKeyCallBacks` and `kCFTypeDictionaryValueCallBacks` (page 218). The dictionary is copied with `CFDictionaryCreateCopy` (page 206). If you do not want the userInfo dictionary, you can pass `NULL`, in which case an empty dictionary will be assigned.

Return Value

A new CFError object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFError.h`

CFErrorCreateWithUserInfoKeysAndValues

Creates a new CFError object using given keys and values to create the user info dictionary.

```
CFErrorRef CFErrorCreateWithUserInfoKeysAndValues (
   CFAllocatorRef allocator,
   CFStringRef domain,
   CFIndex code,
   const void *const *userInfoKeys,
   const void *const *userInfoValues,
   CFIndex numUserInfoValues
);
```

Parameters

allocator

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

domain

A CFString that identifies the error domain. If this reference is `NULL` or is otherwise not a valid CFString, the behavior is undefined.

code

A CFIndex that identifies the error code. The code is interpreted within the context of the error domain.

userInfoKeys

An array of *numUserInfoValues* CFStrings used as keys in creating the userInfo dictionary. The value of this parameter can be `NULL` if *numUserInfoValues* is `0`.

userInfoValues

An array of *numUserInfoValues* CF types used as values in creating the userInfo dictionary. The value of this parameter can be `NULL` if *numUserInfoValues* is `0`.

numUserInfoValues

The number of keys and values in the *userInfoKeys* and *userInfoValues* arrays.

Return Value

A new CFError object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFError.h`

CFErrorGetCode

Returns the error code for a given CFError.

```
CFIndex CFErrorGetCode (
   CFErrorRef err
);
```

Parameters

err

The error to examine. If this is not a valid CFError, the behavior is undefined.

Return Value
The error code of *err*.

Discussion
Note that this function returns the error code for the specified CFError, not an error return for the current call.

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

Declared In
CFError.h

CFErrorGetDomain

Returns the error domain for a given CFError.

```
CFStringRef CFErrorGetDomain (
   CFErrorRef err
);
```

Parameters

err
The error to examine. If this is not a valid CFError, the behavior is undefined.

Return Value
The error domain for *err*. Ownership follows the Get Rule.

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

Declared In
CFError.h

CFErrorGetTypeID

Returns the type identifier for the CFError opaque type.

```
CFTypeID CFErrorGetTypeID (
   void
);
```

Return Value
The type identifier for the CFError opaque type.

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

Declared In
CFError.h

Data Types

CFErrorRef

A reference to a CFError object.

```
typedef struct __CFError * CFErrorRef;
```

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

Declared In
`CFError.h`

Constants

Error domains

These constants define domains for CFError objects.

```
const CFStringRef kCFErrorDomainPOSIX;
const CFStringRef kCFErrorDomainOSStatus;
const CFStringRef kCFErrorDomainMach;
const CFStringRef kCFErrorDomainCocoa;
```

Constants

`kCFErrorDomainPOSIX`
A constant that specified the POSIX domain.
Available in Mac OS X v10.5 and later.
Declared in `CFError.h`.

`kCFErrorDomainOSStatus`
A constant that specified the OS domain.
Available in Mac OS X v10.5 and later.
Declared in `CFError.h`.

`kCFErrorDomainMach`
A constant that specified the Mach domain.
Available in Mac OS X v10.5 and later.
Declared in `CFError.h`.

`kCFErrorDomainCocoa`
A constant that specified the Cocoa domain.
Available in Mac OS X v10.5 and later.
Declared in `CFError.h`.

Discussion
The value of "code" will correspond to preexisting values in these domains.

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

Declared In
`CFError.h`

Keys for the user info dictionary

Keys in the userInfo dictionary.

```
const CFStringRef kCFErrorLocalizedDescriptionKey;
const CFStringRef kCFErrorLocalizedFailureReasonKey;
const CFStringRef kCFErrorLocalizedRecoverySuggestionKey;
const CFStringRef kCFErrorDescriptionKey;
const CFStringRef kCFErrorUnderlyingErrorKey;
```

Constants

`kCFErrorLocalizedDescriptionKey`
Key to identify the end user-presentable description in the userInfo dictionary.

Available in Mac OS X v10.5 and later.

Declared in `CFError.h`.

`kCFErrorLocalizedFailureReasonKey`
Key to identify the end user-presentable failure reason in the userInfo dictionary.

Available in Mac OS X v10.5 and later.

Declared in `CFError.h`.

`kCFErrorLocalizedRecoverySuggestionKey`
Key to identify the end user-presentable recovery suggestion in the userInfo dictionary.

Available in Mac OS X v10.5 and later.

Declared in `CFError.h`.

`kCFErrorDescriptionKey`
Key to identify the description in the userInfo dictionary.

When you create a CFError, you can provide a value for this key if you do not have localizable error strings. The description should be a complete sentence if possible, and should not contain the domain name or error code.

Available in Mac OS X v10.5 and later.

Declared in `CFError.h`.

`kCFErrorUnderlyingErrorKey`
Key to identify the underlying error in the userInfo dictionary.

Available in Mac OS X v10.5 and later.

Declared in `CFError.h`.

Discussion
When you create a user info dictionary, at a minimum you should provide values for one of `kCFErrorLocalizedDescriptionKey` and `kCFErrorLocalizedFailureReasonKey`; ideally you should provide values for `kCFErrorLocalizedDescriptionKey`, `kCFErrorLocalizedFailureReasonKey`, and `kCFErrorLocalizedRecoverySuggestionKey`.

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

Declared In
`NSError.h`

CFFileDescriptor Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFFileDescriptor.h

Overview

The CFFileDescriptor provides an opaque type to monitor file descriptors for read and write activity via CFRunLoop.

You use CFFileDescriptor to monitor file descriptors for read and write activity via CFRunLoop using callbacks. Each call back is one-shot, and must be re-enabled if you want to get another one.

You can re-enable the callback in the callback function itself, but you must completely service the file descriptor before doing so. For example, if you create a CFFileDescriptor for a pipe and get a callback because there are bytes to be read, then if you don't read all of the bytes but nevertheless re-enable the CFFileDescriptor for read activity, you'll get called back again immediately.

You can monitor kqueue file descriptors for read activity to find out when an event the kqueue is filtering for has occurred. You are responsible for understanding the use of the kevent() API and inserting and removing filters from the kqueue file descriptor yourself.

The following example takes a UNIX process ID as argument, and watches up to 20 seconds, and reports if the process terminates in that time:

```
// cc test.c -framework CoreFoundation -O
#include <CoreFoundation/CoreFoundation.h>
#include <unistd.h>
#include <sys/event.h>
static void noteProcDeath(CFFileDescriptorRef fdref, CFOptionFlags callBackTypes,
 void *info) {
    struct kevent kev;
    int fd = CFFileDescriptorGetNativeDescriptor(fdref);
    kevent(fd, NULL, 0, &kev, 1, NULL);
    // take action on death of process here
    printf("process with pid '%u' died\n", (unsigned int)kev.ident);
    CFFileDescriptorInvalidate(fdref);
   CFRelease(fdref); // the CFFileDescriptorRef is no longer of any use in this
 example
}
// one argument, an integer pid to watch, required
int main(int argc, char *argv[]) {
    if (argc < 2) exit(1);
    int fd = kqueue();
    struct kevent kev;
```

```
    EV_SET(&kev, atoi(argv[1]), EVFILT_PROC, EV_ADD|EV_ENABLE, NOTE_EXIT, 0,
NULL);
    kevent(fd, &kev, 1, NULL, 0, NULL);
    CFFileDescriptorRef fdref = CFFileDescriptorCreate(kCFAllocatorDefault, fd,
 true, noteProcDeath, NULL);
    CFFileDescriptorEnableCallBacks(fdref, kCFFileDescriptorReadCallBack);
    CFRunLoopSourceRef source =
CFFileDescriptorCreateRunLoopSource(kCFAllocatorDefault, fdref, 0);
    CFRunLoopAddSource(CFRunLoopGetMain(), source, kCFRunLoopDefaultMode);
    CFRelease(source);
    // run the run loop for 20 seconds
    CFRunLoopRunInMode(kCFRunLoopDefaultMode, 20.0, false);
    return 0;
}
```

Functions by Task

Creating a CFFileDescriptor

`CFFileDescriptorCreate` (page 231)
Creates a new CFFileDescriptor.

Getting Information About a File Descriptor

`CFFileDescriptorGetNativeDescriptor` (page 234)
Returns the native file descriptor for a given CFFileDescriptor.

`CFFileDescriptorIsValid` (page 235)
Returns a Boolean value that indicates whether the native file descriptor for a given CFFileDescriptor is valid.

`CFFileDescriptorGetContext` (page 233)
Gets the context for a given CFFileDescriptor.

Invalidating a File Descriptor

`CFFileDescriptorInvalidate` (page 235)
Invalidates the native file descriptor for a given CFFileDescriptor.

Managing Callbacks

`CFFileDescriptorEnableCallBacks` (page 233)
Enables callbacks for a given CFFileDescriptor.

`CFFileDescriptorDisableCallBacks` (page 232)
Disables callbacks for a given CFFileDescriptor.

Creating a Run Loop Source

`CFFileDescriptorCreateRunLoopSource` (page 232)
Creates a new runloop source for a given CFFileDescriptor.

Getting the CFFileDescriptor Type ID

`CFFileDescriptorGetTypeID` (page 234)
Returns the type identifier for the CFFileDescriptor opaque type.

Functions

CFFileDescriptorCreate

Creates a new CFFileDescriptor.

```
CFFileDescriptorRef CFFileDescriptorCreate (
    CFAllocatorRef allocator,
    CFFileDescriptorNativeDescriptor fd,
    Boolean closeOnInvalidate,
    CFFileDescriptorCallBack callout,
    const CFFileDescriptorContext *context
);
```

Parameters

allocator
The allocator to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

fd
The file descriptor for the new CFFileDescriptor.

closeOnInvalidate
`true` if the new CFFileDescriptor should close *fd* when it is invalidated, otherwise `false`.

callout
The CFFileDescriptorCallBack for the new CFFileDescriptor.

context
Contextual information for the new CFFileDescriptor.

Return Value
A new CFFileDescriptor or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

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

See Also
`CFFileDescriptorGetContext` (page 233)
`CFFileDescriptorInvalidate` (page 235)

Related Sample Code
PreLoginAgents
Watcher

Declared In
`CFFileDescriptor.h`

CFFileDescriptorCreateRunLoopSource

Creates a new runloop source for a given CFFileDescriptor.

```
CFRunLoopSourceRef CFFileDescriptorCreateRunLoopSource (
    CFAllocatorRef allocator,
    CFFileDescriptorRef f,
    CFIndex order
);
```

Parameters

allocator

The allocator to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

f

A CFFileDescriptor.

order

The order for the new run loop (see `CFRunLoopSourceCreate` (page 482)).

Return Value
A new runloop source for *f*, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
The context for the new runloop (see `CFRunLoopSourceCreate` (page 482)) is the same as the context passed in when the CFFileDescriptor was created (see `CFFileDescriptorCreate` (page 231)).

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

Related Sample Code
PreLoginAgents
Watcher

Declared In
`CFFileDescriptor.h`

CFFileDescriptorDisableCallBacks

Disables callbacks for a given CFFileDescriptor.

```
void CFFileDescriptorDisableCallBacks (
    CFFileDescriptorRef f,
    CFOptionFlags callBackTypes
);
```

Parameters

f

A CFFileDescriptor.

callBackTypes

A bitmask that specifies which callbacks to disable (see "Callback Identifiers" (page 237) for possible components).

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

See Also
CFFileDescriptorEnableCallBacks (page 233)

Declared In
CFFileDescriptor.h

CFFileDescriptorEnableCallBacks

Enables callbacks for a given CFFileDescriptor.

```
void CFFileDescriptorEnableCallBacks (
    CFFileDescriptorRef f,
    CFOptionFlags callBackTypes
);
```

Parameters

f

A CFFileDescriptor.

callBackTypes

A bitmask that specifies which callbacks to enable (see "Callback Identifiers" (page 237) for possible components).

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

See Also
CFFileDescriptorDisableCallBacks (page 232)

Related Sample Code
PreLoginAgents
Watcher

Declared In
CFFileDescriptor.h

CFFileDescriptorGetContext

Gets the context for a given CFFileDescriptor.

```
void CFFileDescriptorGetContext (
    CFFileDescriptorRef f,
    CFFileDescriptorContext *context
);
```

Parameters

f

A CFFileDescriptor.

context

Upon return, contains the context passed to *f* in `CFFileDescriptorCreate` (page 231).

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

See Also
`CFFileDescriptorCreate` (page 231)

Declared In
`CFFileDescriptor.h`

CFFileDescriptorGetNativeDescriptor

Returns the native file descriptor for a given CFFileDescriptor.

```
CFFileDescriptorNativeDescriptor CFFileDescriptorGetNativeDescriptor (
    CFFileDescriptorRef f
);
```

Parameters

f

A CFFileDescriptor.

Return Value
The native file descriptor for *f*.

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

See Also
`CFFileDescriptorInvalidate` (page 235)

Declared In
`CFFileDescriptor.h`

CFFileDescriptorGetTypeID

Returns the type identifier for the CFFileDescriptor opaque type.

```
CFTypeID CFFileDescriptorGetTypeID (
    void
);
```

Return Value
The type identifier for the CFFileDescriptor opaque type.

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

Declared In
`CFFileDescriptor.h`

CFFileDescriptorInvalidate

Invalidates the native file descriptor for a given CFFileDescriptor.

```
void CFFileDescriptorInvalidate (
    CFFileDescriptorRef f,
);
```

Parameters

f

A CFFileDescriptor.

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

See Also
`CFFileDescriptorIsValid` (page 235)
`CFFileDescriptorGetNativeDescriptor` (page 234)

Related Sample Code
Watcher

Declared In
`CFFileDescriptor.h`

CFFileDescriptorIsValid

Returns a Boolean value that indicates whether the native file descriptor for a given CFFileDescriptor is valid.

```
Boolean CFFileDescriptorIsValid (
    CFFileDescriptorRef f,
);
```

Parameters

f

A CFFileDescriptor.

Return Value
`true` if the native file descriptor for *f* is valid, otherwise `false`.

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

See Also
`CFFileDescriptorInvalidate` (page 235)

Declared In
`CFFileDescriptor.h`

Data Types

CFFileDescriptorNativeDescriptor

Defines a type for the native file descriptor.

```
typedef int CFFileDescriptorNativeDescriptor;
```

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

Declared In
`CFFileDescriptor.h`

CFFileDescriptorCallBack

Defines a structure for a callback for a CFFileDescriptor.

```
typedef void (*CFFileDescriptorCallBack) (
    CFFileDescriptorRef f,
    CFOptionFlags callBackTypes,
    void *info
);
```

Declared In
`CFFileDescriptor.h`

CFFileDescriptorContext

Defines a structure for the context of a CFFileDescriptor.

```
typedef struct {
    CFIndex    version;
    void *    info;
    void *    (*retain)(void *info);
    void    (*release)(void *info);
    CFStringRef    (*copyDescription)(void *info);
} CFFileDescriptorContext;
```

Fields

`version`
The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is 0.

`info`

`retain`
The retain callback used by the CFFileDescriptor.

`release`
The release callback used by the CFFileDescriptor.

`copyDescription`
The callback used to create a descriptive string representation of the CFFileDescriptor.

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

Declared In
`CFFileDescriptor.h`

CFFileDescriptorRef

A reference to an CFFileDescriptor object.

```
typedef struct __CFFileDescriptor * CFFileDescriptorRef;
```

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

Declared In
`CFFileDescriptor.h`

Constants

Callback Identifiers

Constants that identify the read and write callbacks.

```
enum {
    kCFFileDescriptorReadCallBack = 1 << 0,
    kCFFileDescriptorWriteCallBack = 1 << 1
};
```

Constants

`kCFFileDescriptorReadCallBack`
Identifies the read callback.

Available in Mac OS X v10.5 and later.

Declared in `CFFileDescriptor.h`.

`kCFFileDescriptorWriteCallBack`
Identifies the write callback.

Available in Mac OS X v10.5 and later.

Declared in `CFFileDescriptor.h`.

Declared In
`CFFileDescriptor.h`

CFLocale Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFLocale.h
Companion guides	Locales Programming Guide Internationalization Programming Topics

Overview

Unicode operations such as collation and text boundary determination can be affected by the conventions of a particular language or region. CFLocale objects specify language-specific or region-specific information for locale-sensitive operations.

The CFLocale opaque type provides support for obtaining available locales, obtaining localized locale names, and converting among locale data formats. Locale identifiers in Mac OS X follow the IETF's BCP 47. CFLocale never uses Script Manager codes (except for the legacy support provided by `CFLocaleCreateCanonicalLocaleIdentifierFromScriptManagerCodes` (page 246))—the Script Manager and all its concepts are deprecated.

For more information on locale identifiers and the use of CFLocale, see *Locales Programming Guide*. It is also useful to read the ICU's User Guide for the Locale Class.

CFLocale is "toll-free bridged" with its Cocoa Foundation counterpart, `NSLocale`. 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 `NSLocale *` parameter, you can pass in a `CFLocaleRef`, and in a function where you see a `CFLocaleRef` parameter, you can pass in an `NSLocale` instance. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a Locale

`CFLocaleCopyCurrent` (page 242)
Returns a copy of the logical locale for the current user.

`CFLocaleCreate` (page 245)
Creates a locale for the given arbitrary locale identifier.

`CFLocaleCreateCopy` (page 247)
Returns a copy of a locale.

`CFLocaleGetSystem` (page 250)
Returns the root, canonical locale.

Getting System Locale Information

`CFLocaleCopyAvailableLocaleIdentifiers` (page 241)
Returns an array of CFString objects that represents all locales for which locale data is available.

Getting ISO Information

`CFLocaleCopyISOCountryCodes` (page 243)
Returns an array of CFString objects that represents all known legal ISO country codes.

`CFLocaleCopyISOLanguageCodes` (page 244)
Returns an array of CFString objects that represents all known legal ISO language codes.

`CFLocaleCopyISOCurrencyCodes` (page 244)
Returns an array of CFString objects that represents all known legal ISO currency codes.

`CFLocaleCopyCommonISOCurrencyCodes` (page 242)
Returns an array of strings that represents ISO currency codes for currencies in common use.

Language Preferences

`CFLocaleCopyPreferredLanguages` (page 244)
Returns the array of canonicalized locale IDs that the user prefers.

Getting Information About a Locale

`CFLocaleCopyDisplayNameForPropertyValue` (page 243)
Returns the display name for the given value.

`CFLocaleGetValue` (page 251)
Returns the corresponding value for the given key of a locale's key-value pair.

`CFLocaleGetIdentifier` (page 249)
Returns the given locale's identifier.

Getting and Creating Locale Identifiers

`CFLocaleCreateCanonicalLocaleIdentifierFromScriptManagerCodes` (page 246)
Returns a canonical locale identifier from given language and region codes.

`CFLocaleCreateCanonicalLanguageIdentifierFromString` (page 245)
Returns a canonical language identifier by mapping an arbitrary locale identification string to the canonical identifier

`CFLocaleCreateCanonicalLocaleIdentifierFromString` (page 246)
Returns a canonical locale identifier by mapping an arbitrary locale identification string to the canonical identifier.

`CFLocaleCreateComponentsFromLocaleIdentifier` (page 247)
Returns a dictionary containing the result from parsing a locale ID consisting of language, script, country, variant, and keyword/value pairs.

`CFLocaleCreateLocaleIdentifierFromComponents` (page 248)
Returns a locale identifier consisting of language, script, country, variant, and keyword/value pairs derived from a dictionary containing the source information.

Getting the CFLocale Type ID

`CFLocaleGetTypeID` (page 251)
Returns the type identifier for the CFLocale opaque type.

New Functions

`CFLocaleCreateLocaleIdentifierFromWindowsLocaleCode` (page 249)

`CFLocaleGetLanguageCharacterDirection` (page 250)

`CFLocaleGetLanguageLineDirection` (page 250)

`CFLocaleGetWindowsLocaleCodeFromLocaleIdentifier` (page 252)

Functions

CFLocaleCopyAvailableLocaleIdentifiers

Returns an array of CFString objects that represents all locales for which locale data is available.

```
CFArrayRef CFLocaleCopyAvailableLocaleIdentifiers (
   void
);
```

Return Value
An array of CFString objects that represents all locales for which locale data is available. Ownership follows the Create Rule.

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

Declared In
`CFLocale.h`

CFLocaleCopyCommonISOCurrencyCodes

Returns an array of strings that represents ISO currency codes for currencies in common use.

```
CFArrayRef CFLocaleCopyCommonISOCurrencyCodes (
   void
);
```

Return Value
An array of CFString objects that represents ISO currency codes for currencies in common use. Ownership follows the Create Rule.

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

Declared In
`CFLocale.h`

CFLocaleCopyCurrent

Returns a copy of the logical locale for the current user.

```
CFLocaleRef CFLocaleCopyCurrent (
   void
);
```

Return Value
The logical locale for the current user that is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences. May return a retained cached object, not a new object. Ownership follows the Create Rule.

Discussion
Settings you get from this locale do not change as a user's preferences are changed so that your operations are consistent. Typically you perform some operations on the returned object and then release it. Since the returned object may be cached, you do not need to hold on to it indefinitely.

Note that locale settings are independent of the user's language setting. The language of the current locale may not correspond to the language at the first index in the `AppleLanguages` array from user defaults. For more details, see Locale Concepts in *Locales Programming Guide*; see also `CFLocaleCopyPreferredLanguages` (page 244).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CFFTPSample
CFPrefTopScores
FSMegaInfo

Declared In
`CFLocale.h`

CFLocaleCopyDisplayNameForPropertyValue

Returns the display name for the given value.

```
CFStringRef CFLocaleCopyDisplayNameForPropertyValue (
   CFLocaleRef displayLocale,
   CFStringRef key,
   CFStringRef value
);
```

Parameters

displayLocale

A locale object.

key

A string that identifies the type that *value* is. It must be one of the standard locale property keys (see "Locale Property Keys" (page 252)).

value

The value for which the display name is required.

Return Value

The display name for *value*. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

Note that not all locale property keys have values with display name values.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFLocale.h`

CFLocaleCopyISOCountryCodes

Returns an array of CFString objects that represents all known legal ISO country codes.

```
CFArrayRef CFLocaleCopyISOCountryCodes (
   void
);
```

Return Value

An array of CFString objects that represents all known legal ISO country codes. Ownership follows the Create Rule.

Discussion

Note: many of these will not have any supporting locale data in Mac OS X.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFLocale.h`

CFLocaleCopyISOCurrencyCodes

Returns an array of CFString objects that represents all known legal ISO currency codes.

```
CFArrayRef CFLocaleCopyISOCurrencyCodes (
   void
);
```

Return Value
An array of CFString objects that represents all known legal ISO currency codes.Ownership follows the Create Rule.

Discussion
Note: many of these will not have any supporting locale data in Mac OS X.

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

Declared In
`CFLocale.h`

CFLocaleCopyISOLanguageCodes

Returns an array of CFString objects that represents all known legal ISO language codes.

```
CFArrayRef CFLocaleCopyISOLanguageCodes (
   void
);
```

Return Value
An array of CFString objects that represents all known legal ISO language codes. Ownership follows the Create Rule.

Discussion
Note: many of these will not have any supporting locale data in Mac OS X.

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

Declared In
`CFLocale.h`

CFLocaleCopyPreferredLanguages

Returns the array of canonicalized locale IDs that the user prefers.

```
CFArrayRef CFLocaleCopyPreferredLanguages (
   void
);
```

Return Value
The array of canonicalized CFString locale IDs that the current user prefers. Ownership follows the Create Rule.

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

Declared In
`CFLocale.h`

CFLocaleCreate

Creates a locale for the given arbitrary locale identifier.

```
CFLocaleRef CFLocaleCreate (
   CFAllocatorRef allocator,
   CFStringRef localeIdentifier
);
```

Parameters

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

localeIdentifier
A string representation of an arbitrary locale identifier.

Return Value
A new locale that corresponds to the arbitrary locale identifier *localeIdentifier*. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFLocale.h`

CFLocaleCreateCanonicalLanguageIdentifierFromString

Returns a canonical language identifier by mapping an arbitrary locale identification string to the canonical identifier

```
CFStringRef CFLocaleCreateCanonicalLanguageIdentifierFromString (
   CFAllocatorRef allocator,
   CFStringRef localeIdentifier
);
```

Parameters

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

localeIdentifier
A string representation of an arbitrary locale identifier.

Return Value
A string that represents the canonical language identifier for the specified arbitrary locale identifier. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

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

Declared In
`CFLocale.h`

CFLocaleCreateCanonicalLocaleIdentifierFromScriptManagerCodes

Returns a canonical locale identifier from given language and region codes.

```
CFStringRef CFLocaleCreateCanonicalLocaleIdentifierFromScriptManagerCodes (
   CFAllocatorRef allocator,
   LangCode lcode,
   RegionCode rcode
);
```

Parameters

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

lcode
A Mac OS X language code.

rcode
A Mac OS X region code.

Return Value
A canonical locale identifier created by mapping *lcode* and *rcode* to a locale. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFLocale.h`

CFLocaleCreateCanonicalLocaleIdentifierFromString

Returns a canonical locale identifier by mapping an arbitrary locale identification string to the canonical identifier.

```
CFStringRef CFLocaleCreateCanonicalLocaleIdentifierFromString (
   CFAllocatorRef allocator,
   CFStringRef localeIdentifier
);
```

Parameters

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

localeIdentifier
A string representation of an arbitrary locale identifier (for example, “English”).

Return Value
A canonical locale identifier created by mapping the arbitrary locale identification string to the canonical identifier for the corresponding locale (for example, “en”). Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFLocale.h`

CFLocaleCreateComponentsFromLocaleIdentifier

Returns a dictionary containing the result from parsing a locale ID consisting of language, script, country, variant, and keyword/value pairs.

```
CFDictionaryRef CFLocaleCreateComponentsFromLocaleIdentifier (
   CFAllocatorRef allocator,
   CFStringRef localeID
);
```

Parameters

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

`localeID`
The locale ID to be used when creating the locale dictionary.

Return Value
A dictionary containing the result from parsing a locale ID consisting of language, script, country, variant, and keyword/value pairs. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
The dictionary keys are the constant CFString objects that correspond to the locale ID components; the values correspond to constants where available. For example: the string "en_US@calendar=japanese" yields a dictionary with three entries: `kCFLocaleLanguageCode=en`, `kCFLocaleCountryCode=US`, and `kCFLocaleCalendarIdentifier=kCFJapaneseCalendar`. See also `CFLocaleCreateLocaleIdentifierFromComponents` (page 248).

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

Declared In
`CFLocale.h`

CFLocaleCreateCopy

Returns a copy of a locale.

```
CFLocaleRef CFLocaleCreateCopy (
   CFAllocatorRef allocator,
   CFLocaleRef locale
);
```

Parameters

allocator

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

locale

The locale object to copy.

Return Value

A new locale that is a copy of *locale*. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFLocale.h`

CFLocaleCreateLocaleIdentifierFromComponents

Returns a locale identifier consisting of language, script, country, variant, and keyword/value pairs derived from a dictionary containing the source information.

```
CFStringRef CFLocaleCreateLocaleIdentifierFromComponents (
   CFAllocatorRef allocator,
   CFDictionaryRef dictionary
);
```

Parameters

allocator

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

dictionary

The dictionary to use when creating the locale identifier.

Return Value

A locale identifier consisting of language, script, country, variant, and keyword/value pairs derived from *dictionary*. Returns `NULL` if there was a problem creating the string. Ownership follows the Create Rule.

Discussion

Reverses the actions of `CFLocaleCreateComponentsFromLocaleIdentifier` (page 247), creating a single string from the data in the specified dictionary. For example, the dictionary {`kCFLocaleLanguageCode=en`, `kCFLocaleCountryCode=US`, `kCFLocaleCalendarIdentifier=kCFJapaneseCalendar`} becomes `"en_US@calendar=japanese"`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFLocale.h`

CFLocaleCreateLocaleIdentifierFromWindowsLocaleCode

```
CFStringRef CFLocaleCreateLocaleIdentifierFromWindowsLocaleCode (
   CFAllocatorRef allocator,
   uint32_t lcid
);
```

Parameters

allocator

lcid

Return Value

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFLocale.h

CFLocaleGetIdentifier

Returns the given locale's identifier.

```
CFStringRef CFLocaleGetIdentifier (
   CFLocaleRef locale
);
```

Parameters

locale
The locale object to examine.

Return Value
A string representation of *locale*'s identifier. This may not be the same string that was used to create the locale—it may be canonicalized. Ownership follows the Get Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFLocale.h

CFLocaleGetLanguageCharacterDirection

```
CFLocaleLanguageDirection CFLocaleGetLanguageCharacterDirection (
   CFStringRef isoLangCode
);
```

Parameters

isoLangCode

Return Value

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFLocale.h

CFLocaleGetLanguageLineDirection

```
CFLocaleLanguageDirection CFLocaleGetLanguageLineDirection (
   CFStringRef isoLangCode
);
```

Parameters

isoLangCode

Return Value

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFLocale.h

CFLocaleGetSystem

Returns the root, canonical locale.

```
CFLocaleRef CFLocaleGetSystem (
   void
);
```

Return Value
The root, canonical locale. Ownership follows the Get Rule.

Discussion
The root locale contains fixed backstop settings for all locale information.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LSMSmartCategorizer

Declared In
`CFLocale.h`

CFLocaleGetTypeID

Returns the type identifier for the CFLocale opaque type.

```
CFTypeID CFLocaleGetTypeID (
   void
);
```

Return Value
The type identifier for the CFLocale opaque type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFLocale.h`

CFLocaleGetValue

Returns the corresponding value for the given key of a locale's key-value pair.

```
CFTypeRef CFLocaleGetValue (
   CFLocaleRef locale,
   CFStringRef key
);
```

Parameters

locale
The locale object to examine.

key
The key for which to obtain the corresponding value. Possible values are described in "Locale Property Keys" (page 252).

Return Value
The value corresponding to the given key in locale. The value may be any type of CFType object. Ownership follows the Get Rule.

Discussion
Locale objects use key-value pairs to store property values. Use this function to get the value of a specific property.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFLocale.h`

CFLocaleGetWindowsLocaleCodeFromLocaleIdentifier

```
uint32_t CFLocaleGetWindowsLocaleCodeFromLocaleIdentifier (
   CFStringRef localeIdentifier
);
```

Parameters

localeIdentifier

Return Value

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFLocale.h

Data Types

CFLocaleRef

A reference to a CFLocale object.

```
typedef const struct __CFLocale *CFLocaleRef;
```

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CFLocale.h

Constants

Locale Property Keys

Predefined locale keys used to get property values.

```
const CFStringRef kCFLocaleMeasurementSystem;
const CFStringRef kCFLocaleDecimalSeparator;
const CFStringRef kCFLocaleGroupingSeparator;
const CFStringRef kCFLocaleCurrencySymbol;
const CFStringRef kCFLocaleCurrencyCode;
```

```
const CFStringRef kCFLocaleIdentifier;
const CFStringRef kCFLocaleLanguageCode;
const CFStringRef kCFLocaleCountryCode;
const CFStringRef kCFLocaleScriptCode;
const CFStringRef kCFLocaleVariantCode;
const CFStringRef kCFLocaleExemplarCharacterSet;
const CFStringRef kCFLocaleCalendarIdentifier;
const CFStringRef kCFLocaleCalendar;
const CFStringRef kCFLocaleCollationIdentifier;
const CFStringRef kCFLocaleUsesMetricSystem;
```

Constants

`kCFLocaleMeasurementSystem`

Specifies the measurement system used.

The corresponding value is a CFString, for example “Metric” or “U.S.”.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFLocaleDecimalSeparator`

Specifies the decimal point string.

The corresponding value is a CFString, for example “.” or “,”.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFLocaleGroupingSeparator`

Specifies the separator string between groups of digits.

The corresponding value is a CFString, for example “,” or “.”.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFLocaleCurrencySymbol`

Specifies the currency symbol.

The corresponding value is a CFString, for example “$” or “£”.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFLocaleCurrencyCode`

Specifies the locale currency code.

The corresponding value is a CFString, for example “USD” or “GBP”.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFLocaleIdentifier`

Specifies locale identifier.

The corresponding value is a CFString containing the POSIX locale identifier as used by ICU, such as “`ja_JP`”. If you have a variant locale or a different currency or calendar, it can be as complex as “`en_US_POSIX@calendar=japanese;currency=EUR`” or “`az_Cyrl_AZ@calendar=buddhist;currency=JPY`”.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleLanguageCode`
Specifies the locale language code.

The corresponding value is a CFString containing an ISO 639-x/IETF BCP 47 language identifier, such as "`ja`".

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleCountryCode`
Specifies the locale country code.

The corresponding value is a CFString containing an ISO county code, such as "`JP`".

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleScriptCode`
Specifies the locale script code.

The corresponding value is a CFString containing a Unicode script tag (strictly, an ISO 15924 script tag). Usually this is empty (it is for "`ja_JP`"). It may be present for locales where a script *must* be specified, for example "`uz-Latn-UZ`" vs. "`uz-Cyrl-UZ`" for Uzbek in Latin vs. Cyrillic (in the first case the script code is "`Latn`", and in the second it is "`Cyrl`").

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleVariantCode`
Specifies the locale variant code.

The corresponding value is a CFString containing the variant name. The variant code is arbitrary and application-specific. ICU adds "`_EURO`" to its locale designations for locales that support the Euro currency. For "`en_US_POSIX`" the variant is "`POSIX`", and for "`hy_AM_REVISED`" it is "`REVISED`".

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleExemplarCharacterSet`
Specifies the locale character set.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleCalendarIdentifier`
Specifies the locale calendar identifier.

The corresponding value is a CFString containing the calendar identifier (for possible values, see "Locale Calendar Identifiers" (page 255)).

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleCalendar`
Specifies the locale calendar.

The corresponding value is a CFCalendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleCollationIdentifier`
Specifies the locale collation identifier.

The corresponding value is a collation.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFLocaleUsesMetricSystem`
Specifies the whether the locale uses the metric system.

The corresponding value is a CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

Discussion
Locale objects use key-value pairs to store property values. Use the `CFLocaleGetValue` (page 251) function to get the value of a specific property listed above.

Locale Calendar Identifiers

Predefined locale keys used to get calendar values—values for `kCFLocaleCalendarIdentifier`.

```
const CFStringRef kCFGregorianCalendar;
const CFStringRef kCFBuddhistCalendar;
const CFStringRef kCFChineseCalendar;
const CFStringRef kCFHebrewCalendar;
const CFStringRef kCFIslamicCalendar;
const CFStringRef kCFIslamicCivilCalendar;
const CFStringRef kCFJapaneseCalendar;
const CFStringRef kCFRepublicOfChinaCalendar;
const CFStringRef kCFPersianCalendar;
const CFStringRef kCFIndianCalendar;
const CFStringRef kCFISO8601Calendar;
```

Constants
`kCFGregorianCalendar`
Specifies the Gregorian calendar.

Available in Mac OS X v10.3 and later.

Declared in `CFLocale.h`.

`kCFBuddhistCalendar`
Specifies the Buddhist calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFChineseCalendar`
Specifies the Chinese calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFHebrewCalendar`
Specifies the Hebrew calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFIslamicCalendar`
Specifies the Islamic calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFIslamicCivilCalendar`
Specifies the Islamic Civil calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFJapaneseCalendar`
Specifies the Japanese calendar.

Available in Mac OS X v10.4 and later.

Declared in `CFLocale.h`.

`kCFRepublicOfChinaCalendar`
Specifies the calendar for the Republic of China.

Available in Mac OS X v10.6 and later.

Declared in `CFLocale.h`.

`kCFPersianCalendar`
Specifies the Persian calendar.

Available in Mac OS X v10.6 and later.

Declared in `CFLocale.h`.

`kCFIndianCalendar`
Specifies the Indian calendar.

Available in Mac OS X v10.6 and later.

Declared in `CFLocale.h`.

`kCFISO8601Calendar`
Specifies the ISO 8601 calendar.

Available in Mac OS X v10.6 and later.

Declared in `CFLocale.h`.

Discussion
Locale objects use key-value pairs to store property values. Use the `CFLocaleGetValue` (page 251) function to get the value of a specific property listed above.

Locale Change Notification

Identifier for notification sent if the current locale changes.

```
const CFStringRef kCFLocaleCurrentLocaleDidChangeNotification
```

Constants

`kCFLocaleCurrentLocaleDidChangeNotification`

Identifier for the notification sent if the current locale changes.

This is a local notification posted when the user changes locale information in the System Preferences panel. Keep in mind that there is no order in how notifications are delivered to observers; frameworks or other parts of your code may also be observing this notification to take their own actions, and these may not have occurred at the time you receive the notification.

There is no object or user info for this notification.

Available in Mac OS X v10.5 and later.

Declared in `CFLocale.h`.

Declared In

`CFLocale.h`

CFMachPort Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFMachPort.h

Overview

A CFMachPort object is a wrapper for a native Mach port (`mach_port_t`). Mach ports are the native communication channel for the Mac OS X kernel.

CFMachPort does not provide a function to send messages, so you primarily use a CFMachPort object if you need to listen to a Mach port that you obtained by other means. You can get a callback when a message arrives on the port or when the port becomes invalid, such as when the native port dies.

To listen for messages you need to create a run loop source with `CFMachPortCreateRunLoopSource` (page 261) and add it to a run loop with `CFRunLoopAddSource` (page 457).

Important: If you want to tear down the connection, you must invalidate the port (using `CFMachPortInvalidate` (page 264)) before releasing the runloop source and the Mach port object.

To send data, you must use the Mach APIs with the native Mach port, which is not described here. Alternatively, you can use a *CFMessagePort Reference* object, which can send arbitrary data.

Mach ports only support communication on the local machine. For network communication, you have to use a *CFSocket Reference* object.

Functions by Task

Creating a CFMachPort Object

`CFMachPortCreate` (page 260)
Creates a CFMachPort object with a new Mach port.

`CFMachPortCreateWithPort` (page 262)
Creates a CFMachPort object for a pre-existing native Mach port.

Configuring a CFMachPort Object

`CFMachPortInvalidate` (page 264)
Invalidates a CFMachPort object, stopping it from receiving any more messages.

`CFMachPortCreateRunLoopSource` (page 261)
Creates a CFRunLoopSource object for a CFMachPort object.

`CFMachPortSetInvalidationCallBack` (page 265)
Sets the callback function invoked when a CFMachPort object is invalidated.

Examining a CFMachPort Object

`CFMachPortGetContext` (page 262)
Returns the context information for a CFMachPort object.

`CFMachPortGetInvalidationCallBack` (page 263)
Returns the invalidation callback function for a CFMachPort object.

`CFMachPortGetPort` (page 263)
Returns the native Mach port represented by a CFMachPort object.

`CFMachPortIsValid` (page 265)
Returns a Boolean value that indicates whether a CFMachPort object is valid and able to receive messages.

Getting the CFMachPort Type ID

`CFMachPortGetTypeID` (page 264)
Returns the type identifier for the CFMachPort opaque type.

Functions

CFMachPortCreate

Creates a CFMachPort object with a new Mach port.

```
CFMachPortRef CFMachPortCreate (
   CFAllocatorRef allocator,
   CFMachPortCallBack callout,
   CFMachPortContext *context,
   Boolean *shouldFreeInfo
);
```

Parameters

allocator
The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

callout
The callback function invoked when a message is received on the new Mach port.

context

A structure holding contextual information for the new Mach port. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call.

shouldFreeInfo

A flag set by the function to indicate whether the `info` member of *context* should be freed. The flag is set to `true` on failure, `false` otherwise. *shouldFreeInfo* can be `NULL`.

Return Value
The new CFMachPort object or `NULL` on failure. The CFMachPort object has both send and receive rights. Ownership follows the Create Rule.

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

Declared In
`CFMachPort.h`

CFMachPortCreateRunLoopSource

Creates a CFRunLoopSource object for a CFMachPort object.

```
CFRunLoopSourceRef CFMachPortCreateRunLoopSource (
   CFAllocatorRef allocator,
   CFMachPortRef port,
   CFIndex order
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

port

The Mach port for which to create a CFRunLoopSource object.

order

A priority index indicating the order in which run loop sources are processed. *order* is currently ignored by CFMachPort run loop sources. Pass `0` for this value.

Return Value
The new CFRunLoopSource object for *port*. Ownership follows the Create Rule.

Discussion
The run loop source is not automatically added to a run loop. To add the source to a run loop, use `CFRunLoopAddSource` (page 457).

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

Related Sample Code
NotifyTool

Declared In
`CFMachPort.h`

CFMachPortCreateWithPort

Creates a CFMachPort object for a pre-existing native Mach port.

```
CFMachPortRef CFMachPortCreateWithPort (
   CFAllocatorRef allocator,
   mach_port_t portNum,
   CFMachPortCallBack callout,
   CFMachPortContext *context,
   Boolean *shouldFreeInfo
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

portNum

The native Mach port to use.

callout

The callback function invoked when a message is received on the Mach port.

context

A structure holding contextual information for the Mach port. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call.

shouldFreeInfo

A flag set by the function to indicate whether the `info` member of *context* should be freed. The flag is set to `true` on failure or if a CFMachPort object already exists for *portNum*, `false` otherwise. *shouldFreeInfo* can be `NULL`.

Return Value

The new CFMachPort object or `NULL` on failure. If a CFMachPort object already exists for *portNum*, the function returns the pre-existing object instead of creating a new object; the *context* and *callout* parameters are ignored in this case. Ownership follows the Create Rule.

Discussion

The CFMachPort object does not take full ownership of the send and receive rights of the Mach port *portNum*. It is the caller's responsibility to deallocate the Mach port rights after the CFMachPort object is no longer needed and has been invalidated.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NotifyTool

Declared In

`CFMachPort.h`

CFMachPortGetContext

Returns the context information for a CFMachPort object.

```
void CFMachPortGetContext (
   CFMachPortRef port,
   CFMachPortContext *context
);
```

Parameters

port

The CFMachPort object to examine.

context

A pointer to the structure into which the context information for *port* is to be copied. The information being returned is usually the same information you passed to `CFMachPortCreate` (page 260) or `CFMachPortCreateWithPort` (page 262) when creating *port*. However, if `CFMachPortCreateWithPort` (page 262) returned a cached CFMachPort object instead of creating a new object, *context* is filled with information from the original CFMachPort object instead of the information you passed to the function.

Discussion

The context version number for CFMachPort objects is currently `0`. Before calling this function, you need to initialize the `version` member of *context* to `0`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMachPort.h`

CFMachPortGetInvalidationCallBack

Returns the invalidation callback function for a CFMachPort object.

```
CFMachPortInvalidationCallBack CFMachPortGetInvalidationCallBack (
   CFMachPortRef port
);
```

Parameters

port

The CFMachPort object to examine.

Return Value

The callback function invoked when *port* is invalidated. `NULL` if no callback has been set with `CFMachPortSetInvalidationCallBack` (page 265).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMachPort.h`

CFMachPortGetPort

Returns the native Mach port represented by a CFMachPort object.

```
mach_port_t CFMachPortGetPort (
   CFMachPortRef port
);
```

Parameters

port

The CFMachPort object to examine.

Return Value

The native Mach port represented by *port*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFMachPort.h

CFMachPortGetTypeID

Returns the type identifier for the CFMachPort opaque type.

```
CFTypeID CFMachPortGetTypeID (
   void
);
```

Return Value

The type identifier for the CFMachPort opaque type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFMachPort.h

CFMachPortInvalidate

Invalidates a CFMachPort object, stopping it from receiving any more messages.

```
void CFMachPortInvalidate (
   CFMachPortRef port
);
```

Parameters

port

The CFMachPort object to invalidate.

Discussion

Invalidating a CFMachPort object prevents the port from ever receiving any more messages. The CFMachPort object is not deallocated, though. If the port has not already been invalidated, the port's invalidation callback function is invoked, if one has been set with `CFMachPortSetInvalidationCallBack` (page 265). The `CFMachPortContext` (page 267) `info` information for *port* is also released, if a release callback was specified in the port's context structure. Finally, if a run loop source was created for *port*, the run loop source is invalidated, as well.

If the underlying Mach port is destroyed, the CFMachPort object is automatically invalidated.

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

Declared In
`CFMachPort.h`

CFMachPortIsValid

Returns a Boolean value that indicates whether a CFMachPort object is valid and able to receive messages.

```
Boolean CFMachPortIsValid (
   CFMachPortRef port
);
```

Parameters

port

The CFMachPort object to examine.

Return Value
`true` if *port* can be used for communication, otherwise `false`.

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

Declared In
`CFMachPort.h`

CFMachPortSetInvalidationCallBack

Sets the callback function invoked when a CFMachPort object is invalidated.

```
void CFMachPortSetInvalidationCallBack (
   CFMachPortRef port,
   CFMachPortInvalidationCallBack callout
);
```

Parameters

port

The CFMachPort object to modify.

callout

The callback function to invoke when *port* is invalidated. Pass `NULL` to remove a callback.

Discussion
If *port* is already invalid, *callout* is invoked immediately.

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

Declared In
`CFMachPort.h`

Callbacks

CFMachPortCallBack

Callback invoked to process a message received on a CFMachPort object.

```
typedef void (*CFMachPortCallBack) (
    CFMachPortRef port,
    void *msg,
    CFIndex size,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFMachPortRef port,
    void *msg,
    CFIndex size,
    void *info
);
```

Parameters

port

The CFMachPort object on which the message *msg* was received.

msg

The Mach message received on *port*. The pointer is to a `mach_msg_header_t` structure.

size

Size of the Mach message *msg*, excluding the message trailer.

info

The `info` member of the `CFMachPortContext` (page 267) structure used when creating *port*.

Discussion

You specify this callback when creating a CFMachPort object with either `CFMachPortCreate` (page 260) or `CFMachPortCreateWithPort` (page 262). To receive messages on a CFMachPort object (and have this callback invoked), you must create a run loop source for the port and add it to a run loop.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMachPort.h`

CFMachPortInvalidationCallBack

Callback invoked when a CFMachPort object is invalidated.

```
typedef void (*CFMachPortInvalidationCallBack) (
    CFMachPortRef port,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFMachPortRef port,
    void *info
);
```

Parameters

port

The CFMachPort object that has been invalidated.

info

The `info` member of the `CFMachPortContext` (page 267) structure used when creating *port*.

Discussion

Your callback should free any resources allocated for *port*.

You specify this callback with `CFMachPortSetInvalidationCallBack` (page 265).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMachPort.h`

Data Types

CFMachPortContext

A structure that contains program-defined data and callbacks with which you can configure a CFMachPort object's behavior.

```
struct CFMachPortContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFMachPortContext CFMachPortContext;
```

Fields

`version`

Version number of the structure. Must be `0`.

`info`

An arbitrary pointer to program-defined data, which can be associated with the CFMachPort object at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

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

Declared In
`CFMachPort.h`

CFMachPortRef

A reference to a CFMachPort object.

```
typedef struct __CFMachPort *CFMachPortRef;
```

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

Declared In
`CFMachPort.h`

CFMessagePort Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFMessagePort.h

Overview

CFMessagePort objects provide a communications channel that can transmit arbitrary data between multiple threads or processes on the local machine.

You create a local message port with `CFMessagePortCreateLocal` (page 270) and make it available to other processes by giving it a name, either when you create it or later with `CFMessagePortSetName` (page 277). Other processes then connect to it using `CFMessagePortCreateRemote` (page 271), specifying the name of the port.

To listen for messages, you need to create a run loop source with `CFMessagePortCreateRunLoopSource` (page 272) and add it to a run loop with `CFRunLoopAddSource` (page 457).

Important: If you want to tear down the connection, you must invalidate the port (using `CFMessagePortInvalidate` (page 274)) before releasing the runloop source and the message port object.

Your message port's callback function will be called when a message arrives. To send data, you store the data in a CFData object and call `CFMessagePortSendRequest` (page 276). You can optionally have the function wait for a reply and return the reply in another CFData object.

Message ports only support communication on the local machine. For network communication, you have to use a CFSocket object.

Functions by Task

Creating a CFMessagePort Object

`CFMessagePortCreateLocal` (page 270)
Returns a local CFMessagePort object.

`CFMessagePortCreateRemote` (page 271)
Returns a CFMessagePort object connected to a remote port.

Configuring a CFMessagePort Object

`CFMessagePortCreateRunLoopSource` (page 272)
Creates a CFRunLoopSource object for a CFMessagePort object.

`CFMessagePortSetInvalidationCallBack` (page 277)
Sets the callback function invoked when a CFMessagePort object is invalidated.

`CFMessagePortSetName` (page 277)
Sets the name of a local CFMessagePort object.

Using a Message Port

`CFMessagePortInvalidate` (page 274)
Invalidates a CFMessagePort object, stopping it from receiving or sending any more messages.

`CFMessagePortSendRequest` (page 276)
Sends a message to a remote CFMessagePort object.

Examining a Message Port

`CFMessagePortGetContext` (page 273)
Returns the context information for a CFMessagePort object.

`CFMessagePortGetInvalidationCallBack` (page 273)
Returns the invalidation callback function for a CFMessagePort object.

`CFMessagePortGetName` (page 274)
Returns the name with which a CFMessagePort object is registered.

`CFMessagePortIsRemote` (page 275)
Returns a Boolean value that indicates whether a CFMessagePort object represents a remote port.

`CFMessagePortIsValid` (page 275)
Returns a Boolean value that indicates whether a CFMessagePort object is valid and able to send or receive messages.

Getting the CFMessagePort Type ID

`CFMessagePortGetTypeID` (page 274)
Returns the type identifier for the CFMessagePort opaque type.

Functions

CFMessagePortCreateLocal

Returns a local CFMessagePort object.

```
CFMessagePortRef CFMessagePortCreateLocal (
   CFAllocatorRef allocator,
   CFStringRef name,
   CFMessagePortCallBack callout,
   CFMessagePortContext *context,
   Boolean *shouldFreeInfo
);
```

Parameters

allocator

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

name

The name with which to register the port. *name* can be `NULL`.

callout

The callback function invoked when a message is received on the message port.

context

A structure holding contextual information for the message port. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call.

shouldFreeInfo

A flag set by the function to indicate whether the `info` member of *context* should be freed. The flag is set to `true` on failure or if a local port named *name* already exists, `false` otherwise. *shouldFreeInfo* can be `NULL`.

Return Value

The new CFMessagePort object, or `NULL` on failure. If a local port is already named *name*, the function returns that port instead of creating a new object; the *context* and *callout* parameters are ignored in this case. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

BasicInputMethod

Declared In

`CFMessagePort.h`

CFMessagePortCreateRemote

Returns a CFMessagePort object connected to a remote port.

```
CFMessagePortRef CFMessagePortCreateRemote (
   CFAllocatorRef allocator,
   CFStringRef name
);
```

Parameters

allocator

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

name

The name of the remote message port to which to connect.

Return Value

The new CFMessagePort object, or `NULL` on failure. If a message port has already been created for the remote port, the pre-existing object is returned. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

BasicInputMethod

Declared In

`CFMessagePort.h`

CFMessagePortCreateRunLoopSource

Creates a CFRunLoopSource object for a CFMessagePort object.

```
CFRunLoopSourceRef CFMessagePortCreateRunLoopSource (
   CFAllocatorRef allocator,
   CFMessagePortRef local,
   CFIndex order
);
```

Parameters

allocator

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

ms

The message port for which to create a run loop source.

order

A priority index indicating the order in which run loop sources are processed. *order* is currently ignored by CFMessagePort object run loop sources. Pass `0` for this value.

Return Value

The new CFRunLoopSource object for *ms*. Ownership follows the Create Rule.

Discussion

The run loop source is not automatically added to a run loop. To add the source to a run loop, use `CFRunLoopAddSource` (page 457).

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

Related Sample Code
BackgroundExporter
BasicInputMethod

Declared In
`CFMessagePort.h`

CFMessagePortGetContext

Returns the context information for a CFMessagePort object.

```
void CFMessagePortGetContext (
   CFMessagePortRef ms,
   CFMessagePortContext *context
);
```

Parameters

ms

The message port to examine.

context

A pointer to the structure into which the context information for *ms* is to be copied. The information being returned is usually the same information you passed to `CFMessagePortCreateLocal` (page 270) when creating *ms*. However, if `CFMessagePortCreateLocal` (page 270) returned a cached object instead of creating a new object, *context* is filled with information from the original message port instead of the information you passed to the function.

Discussion
The context version number for message ports is currently `0`. Before calling this function, you need to initialize the `version` member of *context* to `0`.

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

Declared In
`CFMessagePort.h`

CFMessagePortGetInvalidationCallBack

Returns the invalidation callback function for a CFMessagePort object.

```
CFMessagePortInvalidationCallBack CFMessagePortGetInvalidationCallBack (
   CFMessagePortRef ms
);
```

Parameters

ms

The message port to examine.

Return Value
The callback function invoked when *ms* is invalidated. `NULL` if no callback has been set with `CFMessagePortSetInvalidationCallBack` (page 277).

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

Declared In
`CFMessagePort.h`

CFMessagePortGetName

Returns the name with which a CFMessagePort object is registered.

```
CFStringRef CFMessagePortGetName (
   CFMessagePortRef ms
);
```

Parameters

ms
The message port to examine.

Return Value
The registered name of *ms*, `NULL` if unnamed. Ownership follows the Get Rule.

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

Declared In
`CFMessagePort.h`

CFMessagePortGetTypeID

Returns the type identifier for the CFMessagePort opaque type.

```
CFTypeID CFMessagePortGetTypeID (
   void
);
```

Return Value
The type identifier for the CFMessagePort opaque type.

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

Declared In
`CFMessagePort.h`

CFMessagePortInvalidate

Invalidates a CFMessagePort object, stopping it from receiving or sending any more messages.

```
void CFMessagePortInvalidate (
   CFMessagePortRef ms
);
```

Parameters

ms

The message port to invalidate.

Discussion

Invalidating a message port prevents the port from ever sending or receiving any more messages; the message port is not deallocated, though. If the port has not already been invalidated, the port's invalidation callback function is invoked, if one has been set with `CFMessagePortSetInvalidationCallBack` (page 277). The `CFMessagePortContext` (page 279) `info` information for *ms* is also released, if a release callback was specified in the port's context structure. Finally, if a run loop source was created for *ms*, the run loop source is also invalidated.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

Declared In

`CFMessagePort.h`

CFMessagePortIsRemote

Returns a Boolean value that indicates whether a CFMessagePort object represents a remote port.

```
Boolean CFMessagePortIsRemote (
   CFMessagePortRef ms
);
```

Parameters

ms

The message port to examine.

Return Value

`true` if *ms* is a remote port, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMessagePort.h`

CFMessagePortIsValid

Returns a Boolean value that indicates whether a CFMessagePort object is valid and able to send or receive messages.

```
Boolean CFMessagePortIsValid (
   CFMessagePortRef ms
);
```

Parameters

ms

The message port to examine.

Return Value

`true` if *ms* can be used for communication, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFMessagePort.h`

CFMessagePortSendRequest

Sends a message to a remote CFMessagePort object.

```
SInt32 CFMessagePortSendRequest (
   CFMessagePortRef remote,
   SInt32 msgid,
   CFDataRef data,
   CFTimeInterval sendTimeout,
   CFTimeInterval rcvTimeout,
   CFStringRef replyMode,
   CFDataRef *returnData
);
```

Parameters

remote

The message port to which *data* should be sent.

msgid

An arbitrary integer value that you can send with the message.

data

The data to send to *remote*.

sendTimeout

The time to wait for *data* to be sent.

rcvTimeout

The time to wait for a reply to be returned.

replyMode

The run loop mode in which the function should wait for a reply. If the message is a `oneway` (so no response is expected), then *replyMode* should be `NULL`. If *replyMode* is non-`NULL`, the function runs the run loop waiting for a reply, in that mode. *replyMode* can be any string name of a run loop mode, but it should be one with input sources installed. You should use the `kCFRunLoopDefaultMode` constant unless you have a specific reason to use a different mode.

returnData

Upon return, contains a CFData object containing the reply data. Ownership follows the Create Rule.

Return Value
Error code indicating success or failure. See "CFMessagePortSendRequest Error Codes" (page 280) for the possible return values.

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

Related Sample Code
BackgroundExporter
BasicInputMethod

Declared In
`CFMessagePort.h`

CFMessagePortSetInvalidationCallBack

Sets the callback function invoked when a CFMessagePort object is invalidated.

```
void CFMessagePortSetInvalidationCallBack (
   CFMessagePortRef ms,
   CFMessagePortInvalidationCallBack callout
);
```

Parameters

ms
The message port to examine.

callout
The callback function to invoke when *ms* is invalidated. Pass `NULL` to remove a callback.

Discussion
If *ms* is already invalid, *callout* is invoked immediately.

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

Declared In
`CFMessagePort.h`

CFMessagePortSetName

Sets the name of a local CFMessagePort object.

```
Boolean CFMessagePortSetName (
   CFMessagePortRef ms,
   CFStringRef newName
);
```

Parameters

ms
The local message port to examine.

newName
The new name for *ms*.

Return Value
`true` if the name change succeeds, otherwise `false`.

Discussion
Other threads and processes can connect to a named message port with `CFMessagePortCreateRemote` (page 271).

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

Declared In
`CFMessagePort.h`

Callbacks

CFMessagePortCallBack

Callback invoked to process a message received on a CFMessagePort object.

```
typedef CFDataRef (*CFMessagePortCallBack) (
    CFMessagePortRef local,
    SInt32 msgid,
    CFDataRef data,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFDataRef MyCallBack (
    CFMessagePortRef local,
    SInt32 msgid,
    CFDataRef data,
    void *info
);
```

Parameters

local
The local message port that received the message.

msgid
An arbitrary integer value assigned to the message by the sender.

data
The message data.

info
The `info` member of the `CFMessagePortContext` (page 279) structure that was used when creating *local*.

Return Value
Data to send back to the sender of the message. The system releases the returned CFData object. Return `NULL` if you want an empty reply returned to the sender.

Discussion
If you want the message data to persist beyond this callback, you must explicitly create a copy of *data* rather than merely retain it; the contents of *data* will be deallocated after the callback exits.

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

Declared In
CFMessagePort.h

CFMessagePortInvalidationCallBack

Callback invoked when a CFMessagePort object is invalidated.

```
typedef void (*CFMessagePortInvalidationCallBack) (
    CFMessagePortRef ms,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFMessagePortRef ms,
    void *info
);
```

Parameters

ms
The message port that has been invalidated.

info
The `info` member of the `CFMessagePortContext` (page 279) structure that was used when creating *ms*, if *ms* is a local port; `NULL` if *ms* is a remote port.

Discussion
Your callback should free any resources allocated for *ms*.

You specify this callback with `CFMessagePortSetInvalidationCallBack` (page 277).

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

Declared In
CFMessagePort.h

Data Types

CFMessagePortContext

A structure that contains program-defined data and callbacks with which you can configure a CFMessagePort object's behavior.

```
struct CFMessagePortContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFMessagePortContext CFMessagePortContext;
```

Fields

`version`
: Version number of the structure. Must be `0`.

`info`
: An arbitrary pointer to program-defined data, which can be associated with the message port at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
: A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
: A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
: A copy description callback for your program-defined `info` pointer. Can be `NULL`.

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

Declared In
`CFMessagePort.h`

CFMessagePortRef

A reference to a message port object.

```
typedef struct __CFMessagePort *CFMessagePortRef;
```

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

Declared In
`CFMessagePort.h`

Constants

CFMessagePortSendRequest Error Codes

Error codes for `CFMessagePortSendRequest`.

```
enum {
    kCFMessagePortSuccess = 0,
    kCFMessagePortSendTimeout = -1,
    kCFMessagePortReceiveTimeout = -2,
    kCFMessagePortIsInvalid = -3,
    kCFMessagePortTransportError = -4
};
```

Constants

`kCFMessagePortSuccess`

The message was successfully sent and, if a reply was expected, a reply was received.

Available in Mac OS X v10.0 and later.

Declared in `CFMessagePort.h`.

`kCFMessagePortSendTimeout`

The message could not be sent before the send timeout.

Available in Mac OS X v10.0 and later.

Declared in `CFMessagePort.h`.

`kCFMessagePortReceiveTimeout`

No reply was received before the receive timeout.

Available in Mac OS X v10.0 and later.

Declared in `CFMessagePort.h`.

`kCFMessagePortIsInvalid`

The message could not be sent because the message port is invalid.

Available in Mac OS X v10.0 and later.

Declared in `CFMessagePort.h`.

`kCFMessagePortTransportError`

An error occurred trying to send the message.

Available in Mac OS X v10.0 and later.

Declared in `CFMessagePort.h`.

CFMutableArray Reference

Derived From:	*CFArray Reference* : *CFPropertyList Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFArray.h
Companion guides	Collections Programming Topics for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFMutableArray manages dynamic arrays. The basic interface for managing arrays is provided by *CFArray Reference*. CFMutableArray adds functions to modify the contents of an array.

You create a mutable array object using either the `CFArrayCreateMutable` (page 285) or `CFArrayCreateMutableCopy` (page 286) function.

CFMutableArray provides several functions for changing the contents of an array, for example the `CFArrayAppendValue` (page 284) and `CFArrayInsertValueAtIndex` (page 287) functions add values to an array and `CFArrayRemoveValueAtIndex` (page 288) removes values from an array. You can also reorder the contents of an array using `CFArrayExchangeValuesAtIndices` (page 286) and `CFArraySortValues` (page 290).

CFMutableArray is "toll-free bridged" with its Cocoa Foundation counterpart, `NSMutableArray`. 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 `NSMutableArray *` parameter, you can pass in a `CFMutableArrayRef`, and in a function where you see a `CFMutableArrayRef` parameter, you can pass in an NSMutableArray instance. This fact also applies to concrete subclasses of NSMutableArray. See Interchangeable Data Types for more information on toll-free bridging.

Functions

CFArrayAppendArray

Adds the values from one array to another array.

```
void CFArrayAppendArray (
   CFMutableArrayRef theArray,
   CFArrayRef otherArray,
   CFRange otherRange
);
```

Parameters

theArray

The array to which values from *otherArray* are added. If *theArray* is a limited-capacity array, adding `otherRange.length` values from *otherArray* must not cause the capacity limit of *theArray* to be exceeded.

otherArray

An array providing the values to be added to *theArray*.

otherRange

The range within *otherArray* from which to add the values to *theArray*. The range must not exceed the index space of *otherArray*.

Discussion

The new values are retained by *theArray* using the retain callback provided when *theArray* was created. If the values are not of the type expected by the retain callback, the behavior is undefined. The values are assigned to the indices one larger than the previous largest index in *theArray*, and beyond, and the count of *theArray* is increased by `otherRange.length`. The values are assigned new indices in *theArray* from smallest to largest index in the order in which they appear in *otherArray*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFArray.h`

CFArrayAppendValue

Adds a value to an array giving it the new largest index.

```
void CFArrayAppendValue (
   CFMutableArrayRef theArray,
   const void *value
);
```

Parameters

theArray

The array to which *value* is to be added. If *theArray* is a limited-capacity array and it is full before this operation, the behavior is undefined.

value

A CFType object or a pointer value to add to *theArray*.

Discussion

The *value* parameter is retained by *theArray* using the retain callback provided when *theArray* was created. If *value* is not of the type expected by the retain callback, the behavior is undefined. The *value* parameter is assigned to the index one larger than the previous largest index and the count of *theArray* is increased by one.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
HID Config Save
HID Explorer
ImageClient
MoreSCF

Declared In
CFArray.h

CFArrayCreateMutable

Creates a new empty mutable array.

```
CFMutableArrayRef CFArrayCreateMutable (
   CFAllocatorRef allocator,
   CFIndex capacity,
   const CFArrayCallBacks *callBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new array and its storage for values. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

capacity

The maximum number of values that can be contained by the new array. The array starts empty and can grow to this number of values (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. The value must not be negative.

callBacks

A pointer to a CFArrayCallBacks (page 50) structure initialized with the callbacks for the array to use on each value in the array. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple array creations.

If the array contains CFType objects only, then pass kCFTypeArrayCallBacks (page 52) to use the default callback functions.

This parameter may be NULL, which is treated as if a valid structure of version 0 with all fields NULL had been passed in.

If any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a CFArrayCallBacks structure, the behavior is undefined. If any value put into the array is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

Return Value
A new mutable array, 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
BetterAuthorizationSample
HID Calibrator
HID Config Save

HID Explorer
ImageClient

Declared In
CFArray.h

CFArrayCreateMutableCopy

Creates a new mutable array with the values from another array.

```
CFMutableArrayRef CFArrayCreateMutableCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFArrayRef theArray
);
```

Parameters

allocator

The allocator to use to allocate memory for the new array and its storage for values. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

capacity

The maximum number of values that can be contained by the new array. The array starts with the same number of values as *theArray* and can grow to this number of values (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. If non-0, *capacity* must be greater than or equal to the count of *theArray*.

theArray

The array to copy. The pointer values from the array are copied into the new array. However, the values are also retained by the new array.

Return Value
A new mutable array that contains the same values as *theArray*. The new array has the same count as the *theArray* and uses the same callbacks. Ownership follows the Create Rule.

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

Related Sample Code
audioburntest
CFPrefTopScores
databurntest
MFSLives
MoreSCF

Declared In
CFArray.h

CFArrayExchangeValuesAtIndices

Exchanges the values at two indices of an array.

```
void CFArrayExchangeValuesAtIndices (
   CFMutableArrayRef theArray,
   CFIndex idx1,
   CFIndex idx2
);
```

Parameters

theArray

The array that contains the values to be swapped.

idx1

The index of the value to swap with the value at *idx2*. The index must not exceed the index space of *theArray* (`0` to `N-1` inclusive, where `N` is the count of *theArray* before the operation).

idx2

The index of the value to swap with the value at *idx1*. The index must not exceed the index space of *theArray* (`0` to `N-1` inclusive, where `N` is the count of *theArray* before the operation).

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

Declared In
`CFArray.h`

CFArrayInsertValueAtIndex

Inserts a value into an array at a given index.

```
void CFArrayInsertValueAtIndex (
   CFMutableArrayRef theArray,
   CFIndex idx,
   const void *value
);
```

Parameters

theArray

The array into which *value* is inserted. If *theArray* is a fixed-capacity array and it is full before this operation, the behavior is undefined.

idx

The index at which to insert *value*. The index must be in the range `0` to `N` inclusive, where `N` is the count of *theArray* before the operation. If the index is the same as the count of *theArray*, this function has the same effect as `CFArrayAppendValue` (page 284).

value

The value to insert into *theArray*. The value is retained by *theArray* using the retain callback provided when *theArray* was created. If *value* is not of the type expected by the retain callback, the behavior is undefined.

Discussion
The *value* parameter is assigned to the index *idx*, and all values in *theArray* with equal and larger indices have their indices increased by one.

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

Related Sample Code
CFPrefTopScores
CoreTextRTF
MoreSCF
RecentItems

Declared In
`CFArray.h`

CFArrayRemoveAllValues

Removes all the values from an array, making it empty.

```
void CFArrayRemoveAllValues (
   CFMutableArrayRef theArray
);
```

Parameters

theArray
The array from which all of the values are removed.

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

Related Sample Code
ImageClient

Declared In
`CFArray.h`

CFArrayRemoveValueAtIndex

Removes the value at a given index from an array.

```
void CFArrayRemoveValueAtIndex (
   CFMutableArrayRef theArray,
   CFIndex idx
);
```

Parameters

theArray
The array from which the value is to be removed.

idx
The index of the value to remove. The value not lie outside the index space of *theArray* (`0` to `N-1` inclusive, where `N` is the count of *theArray* before the operation).

Discussion
All values in *theArray* with indices larger than *idx* have their indices decreased by one.

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

Related Sample Code
CFLocalServer
CFPrefTopScores
ImageClient
MFSLives
MoreSCF

Declared In
`CFArray.h`

CFArrayReplaceValues

Replaces a range of values in an array.

```
void CFArrayReplaceValues (
   CFMutableArrayRef theArray,
   CFRange range,
   const void **newValues,
   CFIndex newCount
);
```

Parameters

theArray

The array in which some values are to be replaced. If this parameter is not a valid CFMutableArray object, the behavior is undefined.

range

The range of values within *theArray* to replace. The range location or end point (defined by the location plus length minus 1) must not lie outside the index space of *theArray* (`0` to `N-1` inclusive, where `N` is the count of *theArray*). The range length must not be negative. The range may be empty (length 0), in which case the new values are merely inserted at the range location.

newValues

A C array of the pointer-sized values to be placed into *theArray*. The new values in *theArray* are ordered in the same order in which they appear in this C array. This parameter may be `NULL` if the *newCount* parameter is 0. This C array is not changed or freed by this function. If this parameter is not a valid pointer to a C array of at least *newCount* pointers, the behavior is undefined.

newCount

The number of values to copy from the *newValues* C array into *theArray*. If this parameter is different from the range length, the excess *newCount* values are inserted after the range or the excess range values are deleted. This parameter may be 0, in which case no new values are replaced into *theArray* and the values in the range are simply removed. If this parameter is negative or greater than the number of values actually in the *newValues* C array, the behavior is undefined.

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

Related Sample Code
DockBrowser

Declared In
`CFArray.h`

CFArraySetValueAtIndex

Changes the value at a given index in an array.

```
void CFArraySetValueAtIndex (
   CFMutableArrayRef theArray,
   CFIndex idx,
   const void *value
);
```

Parameters

theArray

The array in which the value is to be changed.

idx

The index at which to set the new value. The value must not lie outside the index space of *theArray* (0 to N-1 inclusive, where N is the count of the array before the operation).

value

The value to set in *theArray*. The value is retained by *theArray* using the retain callback provided when *theArray* was created and the previous value at *idx* is released. If the value is not of the type expected by the retain callback, the behavior is undefined. The indices of other values are not affected.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlbumToSlideshow

From A View to A Movie

Watcher

Declared In

CFArray.h

CFArraySortValues

Sorts the values in an array using a given comparison function.

```
void CFArraySortValues (
   CFMutableArrayRef theArray,
   CFRange range,
   CFComparatorFunction comparator,
   void *context
);
```

Parameters

theArray

The array whose values are sorted.

range

The range of values within *theArray* to sort. The range location or end point (defined by the location plus length minus 1) must not lie outside the index space of *theArray* (0 to N-1 inclusive, where N is the count of *theArray*). The range length must not be negative. The range may be empty (length 0).

comparator

The function with the comparator function type signature that is used in the sort operation to compare the values in *theArray*. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. If there are values in *theArray* that the *comparator* function does not expect or cannot properly compare, the behavior is undefined. The values in the range are sorted from least to greatest according to this function.

context

A pointer-sized program-defined value, which is passed as the third parameter to the *comparator* function, but is otherwise unused by this function. If the context is not what is expected by the *comparator* function, the behavior is undefined.

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

Related Sample Code
DockBrowser
HID Dumper
HID Explorer
MFSLives
MoreSCF

Declared In
`CFArray.h`

Data Types

CFMutableArrayRef

A reference to a mutable array object.

```
typedef struct __CFArray *CFMutableArrayRef;
```

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

Declared In
`CFArray.h`

CFMutableAttributedString Reference

Derived From:	*CFPropertyList ReferenceCFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFMutableAttributedString.h CFBase.h
Companion guides	Property List Programming Topics for Core Foundation String Programming Guide for Core Foundation Data Formatting Guide for Core Foundation

Overview

Instances of CFMutableAttributedString manage mutable character strings and associated sets of attributes (for example, font and kerning information) that apply to individual characters or ranges of characters in the string. CFAttributedString as defined in CoreFoundation provides the basic container functionality, while higher levels provide definitions for standard attributes, their values, and additional behaviors involving these. CFMutableAttributedString represents a mutable string—use CFAttributedString to create and manage an attributed string that cannot be changed after it has been created.

iOS Note: APIs to draw attributed strings are only available on iOS 3.2 and later.

CFMutableAttributedString is not a "subclass" of CFMutableString; that is, it does not respond to CFMutableString (or CFString) function calls. CFAttributedString conceptually contains a CFMutableString to which it applies attributes. This protects you from ambiguities caused by the semantic differences between simple and attributed string. Functions defined for CFAttributedString can be applied to a CFMutableAttributedString object.

Attributes are identified by key/value pairs stored in CFDictionary objects. Keys must be CFString objects, while the corresponding values are CFType objects of an appropriate type. See the attribute constants in *NSAttributedString Application Kit Additions Reference* for standard attribute names on Mac OS X.

Important: Attribute dictionaries set for an attributed string must always be created with kCFCopyStringDictionaryKeyCallbacks for their dictionary key callbacks and kCFTypeDictionaryValueCallBacks for their value callbacks; otherwise it's an error.

When you modify the contents of a mutable attributed string, it may have to do a lot of work to ensure it is internally consistent, and to coalesce runs of identical attributes. You can call `CFAttributedStringBeginEditing` (page 295) and `CFAttributedStringEndEditing` (page 296) around a set of related mutation calls that don't require the string to be in consistent state in between, and thereby reduce the amount of work necessary. These calls can be nested.

On Mac OS X, CFMutableAttributedString is "toll-free bridged" with its Cocoa Foundation counterpart, NSMutableAttributedString. 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 `NSMutableAttributedString *` parameter, you can pass in an object of type `CFMutableAttributedStringRef`, and in a function where you see a `CFMutableAttributedStringRef` parameter, you can pass in an `NSMutableAttributedString` instance. See Interchangeable Data Types for more information on toll-free bridging.

iOS Note: `NSMutableAttributedString` is only available on iOS 3.2 and later.

There is not always a 1:1 mapping between `NSMutableAttributedString`'s methods and CFMutableAttributedString's functions. For example, to perform an operation equivalent to `NSMutableAttributedString`'s `appendAttributedString:` method on a CFMutableAttributedString object, you can use `CFAttributedStringReplaceAttributedString` (page 298) and specify `CFRangeMake(CFAttributedStringGetLength(attrStr), 0)` as the range. Alternatively you can cast the CFMutableAttributedString object to an `NSMutableAttributedString` object and send the `appendAttributedString:` message.

Functions by Task

Creating a CFMutableAttributedString

`CFAttributedStringCreateMutable` (page 295)
Creates a mutable attributed string.

`CFAttributedStringCreateMutableCopy` (page 296)
Creates a mutable copy of an attributed string.

Modifying a CFMutableAttributedString

`CFAttributedStringBeginEditing` (page 295)
Defers internal consistency-checking and coalescing for a mutable attributed string.

`CFAttributedStringEndEditing` (page 296)
Re-enables internal consistency-checking and coalescing for a mutable attributed string.

`CFAttributedStringGetMutableString` (page 297)
Gets as a mutable string the string for an attributed string.

`CFAttributedStringRemoveAttribute` (page 297)
Removes the value of a single attribute over a specified range.

`CFAttributedStringReplaceString` (page 298)
Modifies the string of an attributed string.

`CFAttributedStringReplaceAttributedString` (page 298)
Replaces the attributed substring over a range with another attributed string.

`CFAttributedStringSetAttribute` (page 299)
Sets the value of a single attribute over the specified range.

`CFAttributedStringSetAttributes` (page 299)
Sets the value of attributes of a mutable attributed string over a specified range.

Functions

CFAttributedStringBeginEditing

Defers internal consistency-checking and coalescing for a mutable attributed string.

```
void CFAttributedStringBeginEditing (
   CFMutableAttributedStringRef aStr
);
```

Parameters

str
A mutable attributed string that is to be edited.

Discussion

Defers internal consistency-checking and coalescing for a mutable attributed string. You must balance a call to this function with a corresponding `CFAttributedStringEndEditing` (page 296).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CFAttributedString.h`

CFAttributedStringCreateMutable

Creates a mutable attributed string.

```
CFMutableAttributedStringRef CFAttributedStringCreateMutable (
   CFAllocatorRef alloc,
   CFIndex maxLength
);
```

Parameters

alloc
An allocator to be used to allocate memory for the new attributed string. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

maxLength
The limit on the length of the new attributed string. The string starts empty and can grow to this length (it can be shorter).

Pass `0` to specify that the maximum length is not limited. The value must not be negative.

Return Value

A new mutable attributed string. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.4 and later.

Declared In
CFAttributedString.h

CFAttributedStringCreateMutableCopy

Creates a mutable copy of an attributed string.

```
CFMutableAttributedStringRef CFAttributedStringCreateMutableCopy (
   CFAllocatorRef alloc,
   CFIndex maxLength,
   CFAttributedStringRef aStr
);
```

Parameters

alloc

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

maxLength

The limit on the length of the new attributed string. The string starts empty and can grow to this length (it can be shorter).

Pass 0 to specify that the maximum length is not limited. If non-0, *maxLength* must be greater than or equal to the length of *aStr*.

aStr

The attributed string to copy.

Return Value
A mutable copy of *aStr*. Ownership follows the Create Rule.

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

Related Sample Code
CoreTextTest

Declared In
CFAttributedString.h

CFAttributedStringEndEditing

Re-enables internal consistency-checking and coalescing for a mutable attributed string.

```
void CFAttributedStringEndEditing (
   CFMutableAttributedStringRef aStr
);
```

Parameters

str

A mutable attributed string, following a call to CFAttributedStringBeginEditing (page 295).

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

Declared In
CFAttributedString.h

CFAttributedStringGetMutableString

Gets as a mutable string the string for an attributed string.

```
CFMutableStringRef CFAttributedStringGetMutableString (
   CFMutableAttributedStringRef aStr
);
```

Parameters

str

The mutable attributed string from which to retrieve the string.

Return Value
The string for the specified attributed string as a mutable string.

Discussion
This function allows you to edit the character contents of the attributed string as if it were a CFMutableString. Attributes corresponding to the edited range are appropriately modified. If, as a result of the edit, new characters are introduced into the string, they inherit the attributes of the first replaced character from range. If no existing characters are replaced by the edit, the new characters inherit the attributes of the character preceding range if it has any, otherwise of the character following range. If the initial string is empty, the attributes for the new characters are also empty.

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

Declared In
CFAttributedString.h

CFAttributedStringRemoveAttribute

Removes the value of a single attribute over a specified range.

```
void CFAttributedStringRemoveAttribute (
   CFMutableAttributedStringRef aStr,
   CFRange range,
   CFStringRef attrName
);
```

Parameters

str

The mutable attributed string to modify.

range

The range of *aStr* from which to remove the specified attribute. *range* must not exceed the bounds of *aStr*.

attrName

The name of the attribute to remove.

Discussion
It is *not* an error of the specified attribute does not exist over the given range.

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

Declared In
`CFAttributedString.h`

CFAttributedStringReplaceAttributedString

Replaces the attributed substring over a range with another attributed string.

```
void CFAttributedStringReplaceAttributedString (
   CFMutableAttributedStringRef aStr,
   CFRange range,
   CFAttributedStringRef replacement
);
```

Parameters

aStr
The mutable attributed string to modify.

range
The range of *aStr* to be modified. *range* must not specify characters outside the bounds of *aStr*.

replacement
The attributed string to replace the contents of *aStr* in *range*.

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

Declared In
`CFAttributedString.h`

CFAttributedStringReplaceString

Modifies the string of an attributed string.

```
void CFAttributedStringReplaceString (
   CFMutableAttributedStringRef aStr,
   CFRange range,
   CFStringRef replacement
);
```

Parameters

aStr
The mutable attributed string to modify.

range
The range of *aStr* to be modified. *range* must not specify characters outside the bounds of *aStr*.

replacement
The string to replace the existing string in *range*.

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

Declared In
CFAttributedString.h

CFAttributedStringSetAttribute

Sets the value of a single attribute over the specified range.

```
void CFAttributedStringSetAttribute (
   CFMutableAttributedStringRef aStr,
   CFRange range,
   CFStringRef attrName,
   CFTypeRef value
);
```

Parameters

aStr
The mutable attributed string to modify.

range
The range of *aStr* over to which the new attributes apply. *range* must not exceed the bounds of *aStr*.

attrName
The name of the attribute whose value to set.

value
The value of the attribute *attrName* to apply over *range*. This value may not be NULL. If you want to remove an attribute, use CFAttributedStringRemoveAttribute (page 297).

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

Related Sample Code
CoreTextTest

Declared In
CFAttributedString.h

CFAttributedStringSetAttributes

Sets the value of attributes of a mutable attributed string over a specified range.

```
void CFAttributedStringSetAttributes (
   CFMutableAttributedStringRef aStr,
   CFRange range,
   CFDictionaryRef replacement,
   Boolean clearOtherAttributes
);
```

Parameters

aStr
The mutable attributed string to modify.

range
The range of aStr over to which the new attributes apply. *range* must not exceed the bounds of *aStr*.

replacement
A dictionary that contains key-value pairs that specify the new attributes to apply to *range*. The keys must be CFString objects, and the corresponding values must be CFType objects.

clearOtherAttributes
If `false`, existing attributes (that aren't being replaced) are left alone; otherwise they are cleared.

Discussion
Note that after this call, if it is mutable, changes to *replacement* will not affect the contents of the attributed string.

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

Related Sample Code
CoreTextTest

Declared In
`CFAttributedString.h`

Data Types

CFMutableAttributedStringRef

A reference to a CFMutableAttributedString object.

```
typedef struct __CFAttributedString *CFMutableAttributedStringRef;
```

Discussion
The `CFMutableAttributedStringRef` type refers to a mutable object that combines a CFString object with a collection of attributes that specify how the characters in the string should be displayed. CFMutableAttributedString is an opaque type that defines the characteristics and behavior of CFMutableAttributedString objects.

CFMutableAttributedString objects also respond to all functions intended for immutable CFAttributedString objects.

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

Declared In
`CFAttributedString.h`

CFMutableBag Reference

Derived From:	CFBag : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBag.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFMutableBag manages dynamic bags. The basic interface for managing bags is provided by CFBag. CFMutableBag adds functions to modify the contents of a bag.

You create a mutable bag object using either the `CFBagCreateMutable` (page 302) or `CFBagCreateMutableCopy` (page 303) function.

CFMutableBag provides several functions for adding and removing values from a bag. The `CFBagAddValue` (page 302) function adds a value to a bag and `CFBagRemoveValue` (page 304) removes values from a bag.

Functions by Task

Creating a Mutable Bag

`CFBagCreateMutable` (page 302)
Creates a new empty mutable bag.

`CFBagCreateMutableCopy` (page 303)
Creates a new mutable bag with the values from another bag.

Modifying a Mutable Bag

`CFBagAddValue` (page 302)
Adds a value to a mutable bag.

`CFBagRemoveAllValues` (page 304)
Removes all values from a mutable bag.

`CFBagRemoveValue` (page 304)
Removes a value from a mutable bag.

`CFBagReplaceValue` (page 305)
Replaces a value in a mutable bag.

`CFBagSetValue` (page 305)
Sets a value in a mutable bag.

Functions

CFBagAddValue

Adds a value to a mutable bag.

```
void CFBagAddValue (
   CFMutableBagRef theBag,
   const void *value
);
```

Parameters

theBag
The bag to which *value* is added.

value
A CFType object or a pointer value to add to *theBag* (or the value itself, if it fits into the size of a pointer).

Discussion

The *value* parameter is retained by *theBag* using the retain callback provided when *theBag* was created. If *value* is not of the type expected by the retain callback, the behavior is undefined. If *value* already exists in the collection, it is simply retained again—no memory is allocated for the added value. Use a CFSet object if you don't want duplicate values in your collection.

Availability

Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In

`CFBag.h`

CFBagCreateMutable

Creates a new empty mutable bag.

```
CFMutableBagRef CFBagCreateMutable (
   CFAllocatorRef allocator,
   CFIndex capacity,
   const CFBagCallBacks *callBacks
);
```

Parameters

allocator
The allocator object to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

capacity

The maximum number of values that can be contained by the new bag. The bag starts empty and can grow to this number of values (and it can have less). If this parameter is `0`, the bag's maximum capacity is not limited. This value must not be negative.

callBacks

A pointer to a `CFBagCallBacks` structure initialized with the callbacks to use to retain, release, describe, and compare values in the bag. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations. This parameter may be `NULL`, which is treated as if a valid structure of version `0` with all fields `NULL` had been passed in.

If any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a `CFBagCallBacks` structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

If the collection contains only CFType objects, then pass `kCFTypeBagCallBacks` as this parameter to use the default callback functions.

Return Value
A new mutable bag, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an new empty mutable bag to which you can add values using the `CFBagAddValue` (page 302) function. The `capacity` parameter specifies the maximum number of values that the CFBag object can contain. If it is `0`, then there is no limit to the number of values that can be added (aside from constraints such as available memory).

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFBag.h`

CFBagCreateMutableCopy

Creates a new mutable bag with the values from another bag.

```
CFMutableBagRef CFBagCreateMutableCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFBagRef theBag
);
```

Parameters

allocator

The allocator to use to allocate memory for the new bag and its storage for values. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

capacity

The maximum number of values that can be contained by the new bag. The bag starts with the same count as *theBag*, and can grow to this number of values (and it can have less). If this value is `0`, the bag's maximum capacity is not limited. This value must be greater than or equal to the count of *theBag*, and must not be negative.

theBag

The bag to copy. The pointer values from *theBag* are copied into the new bag. However, the values are also retained by the new bag. The count of the new bag is the same as the count of *theBag*. The new bag uses the same callbacks as *theBag*.

Return Value
A new mutable bag that contains the same values as *theBag*. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFBag.h

CFBagRemoveAllValues

Removes all values from a mutable bag.

```
void CFBagRemoveAllValues (
   CFMutableBagRef theBag
);
```

Parameters

theBag

The bag from which all of the values are to be removed.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFBag.h

CFBagRemoveValue

Removes a value from a mutable bag.

```
void CFBagRemoveValue (
   CFMutableBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag from which *value* is to be removed.

value

The value to be removed from the collection.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFBag.h

CFBagReplaceValue

Replaces a value in a mutable bag.

```
void CFBagReplaceValue (
   CFMutableBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag from which *value* is to be replaced.

value

The value to be replaced in the collection. If this value does not already exist in the collection, the function does nothing. You may pass the value itself instead of a pointer if it is pointer-size or less. The equal callback provided when *theBag* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theBag*, is not understood by the equal callback, the behavior is undefined.

Discussion
Depending on the implementation of the equal callback specified when creating *theBag*, the object that is replaced by *value* may not have the same pointer equality.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFBag.h

CFBagSetValue

Sets a value in a mutable bag.

```
void CFBagSetValue (
   CFMutableBagRef theBag,
   const void *value
);
```

Parameters

theBag

The bag in which *value* is to be set.

value

The value to be set in the collection. If this value already exists in *theBag*, it is replaced. You may pass the value itself instead of a pointer to it if the value is pointer-size or less. If *theBag* is fixed-size and the value is beyond its capacity, the behavior is undefined.

Discussion
Depending on the implementation of the equal callback specified when creating *theBag*, the value that is replaced by *value* may not have the same pointer equality.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFBag.h`

Data Types

CFMutableBagRef

A reference to a mutable bag object.

```
typedef struct __CFBag *CFMutableBagRef;
```

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

Declared In
`CFBag.h`

CFMutableBitVector Reference

Derived From:	*CFBitVector Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBitVector.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFMutableBitVector objects manage dynamic bit vectors. The basic interface for managing bit vectors is provided by *CFBitVector Reference*. CFMutableBitVector adds functions to modify the contents of a bit vector.

You create a mutable bit vector object using either the `CFBitVectorCreateMutable` (page 308) or `CFBitVectorCreateMutableCopy` (page 308) function. You add to and remove from a bit vector by altering the size of the bit vector with the `CFBitVectorSetCount` (page 312) function

Functions by Task

Creating a CFMutableBitVector Object

`CFBitVectorCreateMutable` (page 308)
Creates a mutable bit vector.

`CFBitVectorCreateMutableCopy` (page 308)
Creates a new mutable bit vector from a pre-existing bit vector.

Modifying a Bit Vector

`CFBitVectorFlipBitAtIndex` (page 309)
Flips a bit value in a bit vector.

`CFBitVectorFlipBits` (page 310)
Flips a range of bit values in a bit vector.

`CFBitVectorSetAllBits` (page 310)
Sets all bits in a bit vector to a particular value.

`CFBitVectorSetBitAtIndex` (page 310)
Sets the value of a particular bit in a bit vector.

`CFBitVectorSetBits` (page 311)
Sets a range of bits in a bit vector to a particular value.

`CFBitVectorSetCount` (page 312)
Changes the size of a mutable bit vector.

Functions

CFBitVectorCreateMutable

Creates a mutable bit vector.

```
CFMutableBitVectorRef CFBitVectorCreateMutable (
   CFAllocatorRef allocator,
   CFIndex capacity
);
```

Parameters

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

capacity
The maximum number of values that can be contained by the new bit vector. The bit vector starts empty and can grow to this number of values (and it can have less).

Pass `0` to specify that the maximum capacity is not limited. The value must not be negative.

Return Value
A new bit vector. Ownership follows the Create Rule.

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

See Also
`CFBitVectorSetCount` (page 312)

Declared In
`CFBitVector.h`

CFBitVectorCreateMutableCopy

Creates a new mutable bit vector from a pre-existing bit vector.

```
CFMutableBitVectorRef CFBitVectorCreateMutableCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFBitVectorRef bv
);
```

Parameters

allocator

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

capacity

The maximum number of values that can be contained by the new bit vector. The bit vector starts with the same number of values as *bv* and can grow to this number of values (it can have less).

Pass `0` to specify that the maximum capacity is not limited. If non-`0`, *capacity* must be large enough to hold all bit values from *bv*.

bv

The bit vector to copy.

Return Value
A new bit vector holding the same bit values as *bv*. Ownership follows the Create Rule

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

See Also
`CFBitVectorSetCount` (page 312)

Declared In
`CFBitVector.h`

CFBitVectorFlipBitAtIndex

Flips a bit value in a bit vector.

```
void CFBitVectorFlipBitAtIndex (
   CFMutableBitVectorRef bv,
   CFIndex idx
);
```

Parameters

bv

The bit vector to modify.

idx

The index of the bit value to flip. The index must be in the range `0`...`N-1`, where `N` is the count of the vector.

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

See Also
`CFBitVectorGetCount` (page 92)

Declared In
CFBitVector.h

CFBitVectorFlipBits

Flips a range of bit values in a bit vector.

```
void CFBitVectorFlipBits (
   CFMutableBitVectorRef bv,
   CFRange range
);
```

Parameters

bv

The bit vector to modify.

range

The range of bit values in *bv* to flip. The range must not exceed 0...N-1, where N is the count of the vector.

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

See Also
CFBitVectorGetCount (page 92)

Declared In
CFBitVector.h

CFBitVectorSetAllBits

Sets all bits in a bit vector to a particular value.

```
void CFBitVectorSetAllBits (
   CFMutableBitVectorRef bv,
   CFBit value
);
```

Parameters

bv

The bit vector to modify.

value

The bit value to which to set all bits in *bv*.

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

Declared In
CFBitVector.h

CFBitVectorSetBitAtIndex

Sets the value of a particular bit in a bit vector.

```
void CFBitVectorSetBitAtIndex (
   CFMutableBitVectorRef bv,
   CFIndex idx,
   CFBit value
);
```

Parameters

bv

The bit vector to modify.

idx

The index of the bit value to set. The index must be in the range `0`…`N-1`, where `N` is the count of the vector.

value

The bit value to which to set the bit at index *idx*.

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

See Also
`CFBitVectorGetCount` (page 92)

Declared In
`CFBitVector.h`

CFBitVectorSetBits

Sets a range of bits in a bit vector to a particular value.

```
void CFBitVectorSetBits (
   CFMutableBitVectorRef bv,
   CFRange range,
   CFBit value
);
```

Parameters

bv

The bit vector to modify.

range

The range of bits to set. The range must not exceed `0`…`N-1`, where `N` is the count of the vector.

value

The bit value to which to set the range of bits.

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

See Also
`CFBitVectorGetCount` (page 92)

Declared In
`CFBitVector.h`

CFBitVectorSetCount

Changes the size of a mutable bit vector.

```
void CFBitVectorSetCount (
   CFMutableBitVectorRef bv,
   CFIndex count
);
```

Parameters

bv

The bit vector to modify.

count

The new size for *bv*. If *count* is greater than the current size of *bv*, the additional bit values are set to 0.

Discussion

If *bv* was created with a fixed capacity, you cannot increase its size beyond that capacity.

Availability

Available in Mac OS X v10.0 and later.

See Also

CFBitVectorGetCount (page 92) (CFBitVector)
CFBitVectorCreateMutable (page 308)
CFBitVectorCreateMutableCopy (page 308)

Declared In

CFBitVector.h

Data Types

CFMutableBitVectorRef

A reference to a mutable bit vector object.

```
typedef struct __CFBitVector *CFMutableBitVectorRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFBitVector.h

CFMutableCharacterSet Reference

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

Overview

CFMutableCharacterSet manages dynamic character sets. The basic interface for managing character sets is provided by CFCharacterSet. CFMutableCharacterSet adds functions to modify the contents of a character set.

You create a mutable character set object using either the `CFCharacterSetCreateMutable` (page 315) or `CFCharacterSetCreateMutableCopy` (page 315) function.

CFMutableCharacterSet is "toll-free bridged" with its Cocoa Foundation counterpart, NSMutableCharacterSet. 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 `NSMutableCharacterSet *` parameter, you can pass in a `CFMutableCharacterSetRef`, and in a function where you see a `CFMutableCharacterSetRef` parameter, you can pass in an NSMutableCharacterSet instance. This capability also applies to concrete subclasses of NSMutableCharacterSet. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a Mutable Character Set

`CFCharacterSetCreateMutable` (page 315)
Creates a new empty mutable character set.

`CFCharacterSetCreateMutableCopy` (page 315)
Creates a new mutable character set with the values from another character set.

Adding Characters

`CFCharacterSetAddCharactersInRange` (page 314)
Adds a given range to a character set.

`CFCharacterSetAddCharactersInString` (page 315)
Adds the characters in a given string to a character set.

Removing Characters

`CFCharacterSetRemoveCharactersInRange` (page 317)
Removes a given range of Unicode characters from a character set.

`CFCharacterSetRemoveCharactersInString` (page 317)
Removes the characters in a given string from a character set.

Logical Operations

`CFCharacterSetIntersect` (page 316)
Forms an intersection of two character sets.

`CFCharacterSetInvert` (page 316)
Inverts the content of a given character set.

`CFCharacterSetUnion` (page 318)
Forms the union of two character sets.

Functions

CFCharacterSetAddCharactersInRange

Adds a given range to a character set.

```
void CFCharacterSetAddCharactersInRange (
   CFMutableCharacterSetRef theSet,
   CFRange theRange
);
```

Parameters

theSet
The character set to modify.

theRange
The range to add to the character set. The range is specified in 32-bits in UTF-32 format, and must lie within the valid Unicode character range (from `0x00000` to `0x10FFFF`).

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetAddCharactersInString

Adds the characters in a given string to a character set.

```
void CFCharacterSetAddCharactersInString (
   CFMutableCharacterSetRef theSet,
   CFStringRef theString
);
```

Parameters

theSet

The character set to modify.

theString

A string containing the characters to add to *theSet*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetCreateMutable

Creates a new empty mutable character set.

```
CFMutableCharacterSetRef CFCharacterSetCreateMutable (
   CFAllocatorRef alloc
);
```

Parameters

alloc

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

Return Value
A new empty mutable character set. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetCreateMutableCopy

Creates a new mutable character set with the values from another character set.

```
CFMutableCharacterSetRef CFCharacterSetCreateMutableCopy (
   CFAllocatorRef alloc,
   CFCharacterSetRef theSet
);
```

Parameters

alloc

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

theSet

The character set to copy.

Return Value

A new mutable character set containing the same characters as *theSet*. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetIntersect

Forms an intersection of two character sets.

```
void CFCharacterSetIntersect (
   CFMutableCharacterSetRef theSet,
   CFCharacterSetRef theOtherSet
);
```

Parameters

theSet

The source character set, modified by intersection with *theOtherSet*.

theOtherSet

The character set with which the intersection is formed.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFCharacterSet.h`

CFCharacterSetInvert

Inverts the content of a given character set.

```
void CFCharacterSetInvert (
   CFMutableCharacterSetRef theSet
);
```

Parameters

theSet

The character set to invert.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFCharacterSet.h

CFCharacterSetRemoveCharactersInRange

Removes a given range of Unicode characters from a character set.

```
void CFCharacterSetRemoveCharactersInRange (
   CFMutableCharacterSetRef theSet,
   CFRange theRange
);
```

Parameters

theSet

The character set to modify.

theRange

The range to remove from the character set. The range is specified in 32-bits in UTF-32 format, and must lie within the valid Unicode character range (from `0x00000` to `0x10FFFF`).

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFCharacterSet.h

CFCharacterSetRemoveCharactersInString

Removes the characters in a given string from a character set.

```
void CFCharacterSetRemoveCharactersInString (
   CFMutableCharacterSetRef theSet,
   CFStringRef theString
);
```

Parameters

theSet

The character set to modify.

theString

A string containing the characters to remove from *theSet*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

CFCharacterSetUnion

Forms the union of two character sets.

```
void CFCharacterSetUnion (
   CFMutableCharacterSetRef theSet,
   CFCharacterSetRef theOtherSet
);
```

Parameters

theSet
The source character set, modified by union with *theOtherSet*.

theOtherSet
The character set with which the union is formed.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFCharacterSet.h`

Data Types

CFMutableCharacterSetRef

A reference to a mutable character set object.

```
typedef struct __CFCharacterSet *CFMutableCharacterSetRef;
```

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

Declared In
`CFCharacterSet.h`

CFMutableData Reference

Derived From:	CFData : CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFData.h
Companion guides	Binary Data Programming Guide for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFMutableData manages dynamic binary data. The basic interface for managing binary data is provided by CFData. CFMutableData adds functions to modify the contents of a binary data object.

You create a mutable data object using either the `CFDataCreateMutable` (page 321) or `CFDataCreateMutableCopy` (page 321) function.

Bytes are added to a data object with the `CFDataAppendBytes` (page 320) function. Bytes are removed from a data object with the `CFDataDeleteBytes` (page 322) function.

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

Functions by Task

Creating a Mutable Data Object

`CFDataCreateMutable` (page 321)
Creates an empty CFMutableData object.

`CFDataCreateMutableCopy` (page 321)
Creates a CFMutableData object by copying another CFData object.

Accessing Data

`CFDataGetMutableBytePtr` (page 323)
Returns a pointer to a mutable byte buffer of a CFMutableData object.

Modifying a Mutable Data Object

`CFDataAppendBytes` (page 320)
Appends the bytes from a byte buffer to the contents of a CFData object.

`CFDataDeleteBytes` (page 322)
Deletes the bytes in a CFMutableData object within a specified range.

`CFDataReplaceBytes` (page 324)
Replaces those bytes in a CFMutableData object that fall within a specified range with other bytes.

`CFDataIncreaseLength` (page 323)
Increases the length of a CFMutableData object's internal byte buffer, zero-filling the extension to the buffer.

`CFDataSetLength` (page 324)
Resets the length of a CFMutableData object's internal byte buffer.

Functions

CFDataAppendBytes

Appends the bytes from a byte buffer to the contents of a CFData object.

```
void CFDataAppendBytes (
    CFMutableDataRef theData,
    const UInt8 *bytes,
    CFIndex length
);
```

Parameters

theData
A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

bytes
A pointer to the buffer of bytes to be added to *theData*.

length
The number of bytes in the byte buffer *bytes*.

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

Related Sample Code
CFLocalServer
ImageClient

Declared In
CFData.h

CFDataCreateMutable

Creates an empty CFMutableData object.

```
CFMutableDataRef CFDataCreateMutable (
    CFAllocatorRef allocator,
    CFIndex capacity
);
```

Parameters

allocator

The CFAllocator object to be used to allocate memory for the new object. Pass NULL or kCFAllocatorDefault to use the current default allocator.

capacity

The maximum number of bytes that the CFData object can contain. The CFData object starts empty and can grow to contain this number of values (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. The value must not be negative.

Return Value
A CFMutableData object or NULL if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an empty (that is, content-less) CFMutableData object. You can add raw data to this object with the CFDataAppendBytes (page 320) function, and thereafter you can replace and delete characters with the appropriate CFMutableData functions. If the *capacity* parameter is greater than 0, any attempt to add characters beyond this limit can result in undefined behavior.

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

Related Sample Code
CFLocalServer
FunHouse
ImageClient
SeeMyFriends
UnsharpMask

Declared In
CFData.h

CFDataCreateMutableCopy

Creates a CFMutableData object by copying another CFData object.

```
CFMutableDataRef CFDataCreateMutableCopy (
    CFAllocatorRef allocator,
    CFIndex capacity,
    CFDataRef theData
);
```

Parameters

allocator

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

capacity

The maximum number of bytes that the CFData object can contain. The CFData object starts with the same length as the original object, and can grow to contain this number of bytes.

Pass `0` to specify that the maximum capacity is not limited. If non-`0`, *capacity* must be greater than or equal to the length of *theData*.

theData

The CFData object to be copied.

Return Value
A CFMutableData object that has the same contents as the original object. Returns `NULL` if there was a problem copying the object. Ownership follows the Create Rule.

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

Related Sample Code
NullAuthPlugin

Declared In
`CFData.h`

CFDataDeleteBytes

Deletes the bytes in a CFMutableData object within a specified range.

```
void CFDataDeleteBytes (
    CFMutableDataRef theData,
    CFRange range
);
```

Parameters

theData

A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

range

The range of bytes (that is, the starting byte and the number of bytes from that point) to delete from *theData*'s byte buffer.

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

Related Sample Code
CFLocalServer

Declared In
CFData.h

CFDataGetMutableBytePtr

Returns a pointer to a mutable byte buffer of a CFMutableData object.

```
UInt8 *CFDataGetMutableBytePtr (
    CFMutableDataRef theData
);
```

Parameters

theData
A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

Return Value
A pointer to the bytes associated with *theData*.

Discussion
If the length of *theData*'s data is not zero, this function is guaranteed to return a pointer to a CFMutableData object's internal bytes. If the length of *theData*'s data *is* zero, this function may or may not return NULL dependent upon many factors related to how the object was created (moreover, in this case the function result might change between different releases and on different platforms).

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

Related Sample Code
NullAuthPlugin

Declared In
CFData.h

CFDataIncreaseLength

Increases the length of a CFMutableData object's internal byte buffer, zero-filling the extension to the buffer.

```
void CFDataIncreaseLength (
    CFMutableDataRef theData,
    CFIndex extraLength
);
```

Parameters

theData
A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

extraLength
The number of bytes by which to increase the byte buffer.

Discussion
This function increases the length of a CFMutableData object's underlying byte buffer to a new size, initializing the new bytes to 0.

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

Declared In
CFData.h

CFDataReplaceBytes

Replaces those bytes in a CFMutableData object that fall within a specified range with other bytes.

```
void CFDataReplaceBytes (
    CFMutableDataRef theData,
    CFRange range,
    const UInt8 *newBytes,
    CFIndex newLength
);
```

Parameters

theData
A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

range
The range of bytes (that is, the starting byte and the number of bytes from that point) to delete from *theData*'s byte buffer.

newBytes
A pointer to the buffer containing the replacement bytes.

newLength
The number of bytes in the byte buffer *newBytes*.

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

Declared In
CFData.h

CFDataSetLength

Resets the length of a CFMutableData object's internal byte buffer.

```
void CFDataSetLength (
    CFMutableDataRef theData,
    CFIndex length
);
```

Parameters

theData
A CFMutableData object. If you pass an immutable CFData object, the behavior is not defined.

length
The new size of *theData*'s byte buffer.

Discussion
This function resets the length of a CFMutableData object's underlying byte buffer to a new size. If that size is less than the current size, it truncates the excess bytes. If that size is greater than the current size, it zero-fills the extension to the byte buffer.

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

Declared In
`CFData.h`

Data Types

CFMutableDataRef

A reference to a CFMutableData object.

```
typedef struct __CFData *CFMutableDataRef;
```

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

Declared In
`CFData.h`

CFMutableDictionary Reference

Derived From:	*CFDictionary Reference* : *CFPropertyList Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFDictionary.h
Companion guides	Collections Programming Topics for Core Foundation Property List Programming Topics for Core Foundation

Overview

CFMutableDictionary manages dynamic dictionaries. The basic interface for managing dictionaries is provided by *CFDictionary Reference*. CFMutableDictionary adds functions to modify the contents of a dictionary.

You create a mutable dictionary object using either the `CFDictionaryCreateMutable` (page 329) or `CFDictionaryCreateMutableCopy` (page 330) function. You can add key-value pairs using the `CFDictionaryAddValue` (page 328) and `CFDictionarySetValue` (page 332) functions. When adding key-value pairs to a dictionary, the keys and values are not copied—they are retained so they are not invalidated before the dictionary is deallocated. You can remove key-value pairs using the `CFDictionaryRemoveValue` (page 331) function. When removing key-value pairs from a dictionary, the keys and values are released.

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

Functions by Task

Creating a Mutable Dictionary

`CFDictionaryCreateMutable` (page 329)
Creates a new mutable dictionary.

`CFDictionaryCreateMutableCopy` (page 330)
Creates a new mutable dictionary with the key-value pairs from another dictionary.

Modifying a Dictionary

`CFDictionaryAddValue` (page 328)
Adds a key-value pair to a dictionary if the specified key is not already present.

`CFDictionaryRemoveAllValues` (page 331)
Removes all the key-value pairs from a dictionary, making it empty.

`CFDictionaryRemoveValue` (page 331)
Removes a key-value pair.

`CFDictionaryReplaceValue` (page 332)
Replaces a value corresponding to a given key.

`CFDictionarySetValue` (page 332)
Sets the value corresponding to a given key.

Functions

CFDictionaryAddValue

Adds a key-value pair to a dictionary if the specified key is not already present.

```
void CFDictionaryAddValue (
   CFMutableDictionaryRef theDict,
   const void *key,
   const void *value
);
```

Parameters

theDict
The dictionary to modify. If the dictionary is a fixed-capacity dictionary and it is full before this operation, the behavior is undefined.

key
The key for the value to add to the dictionary—a CFType object or a pointer value. The *key* is retained by the dictionary using the retain callback provided when the dictionary was created, so must be of the type expected by the callback. If a key which matches *key* is already present in the dictionary, this function does nothing ("add if absent").

value
A CFType object or a pointer value to add to the dictionary. The *value* is retained by the dictionary using the retain callback provided when the dictionary was created, so must be of the type expected by the callback.

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

Related Sample Code
audioburntest
HID Config Save
HID Dumper
MoreSCF
SeeMyFriends

Declared In
CFDictionary.h

CFDictionaryCreateMutable

Creates a new mutable dictionary.

```
CFMutableDictionaryRef CFDictionaryCreateMutable (
   CFAllocatorRef allocator,
   CFIndex capacity,
   const CFDictionaryKeyCallBacks *keyCallBacks,
   const CFDictionaryValueCallBacks *valueCallBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new dictionary and its storage for key-value pairs. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

capacity

The maximum number of key-value pairs that can be contained by the new dictionary. The dictionary starts empty and can grow to this number of key-value pairs (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. The value must not be negative.

keyCallBacks

A pointer to a CFDictionaryKeyCallBacks (page 215) structure initialized with the callbacks to use to retain, release, describe, and compare keys in the dictionary. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be NULL, which is treated as a valid structure of version 0 with all fields NULL. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this value is not a valid pointer to a CFDictionaryKeyCallBacks structure, the behavior is undefined. If any of the keys put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

If the dictionary will contain only CFType objects, then pass a pointer to kCFTypeDictionaryKeyCallBacks (page 218) as this parameter to use the default callback functions.

valueCallBacks

A pointer to a CFDictionaryValueCallBacks (page 216) structure initialized with the callbacks to use to retain, release, describe, and compare values in the dictionary. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be NULL, which is treated as a valid structure of version 0 with all fields NULL. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this value is not a valid pointer to a CFDictionaryValueCallBacks structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

If the dictionary will contain CFType objects only, then pass a pointer to kCFTypeDictionaryValueCallBacks (page 218) as this parameter to use the default callback functions.

Return Value
A new dictionary, or NULL if there was a problem creating the object. Ownership follows the Create Rule in *Memory Management Programming Guide for Core Foundation*.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Dumper
HID Utilities
ImageClient

Declared In
`CFDictionary.h`

CFDictionaryCreateMutableCopy

Creates a new mutable dictionary with the key-value pairs from another dictionary.

```
CFMutableDictionaryRef CFDictionaryCreateMutableCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFDictionaryRef theDict
);
```

Parameters

allocator

The allocator to use to allocate memory for the new dictionary and its storage for key-value pairs. Pass `NULL` or `kCFAllocatorDefault` (page 35) to use the current default allocator.

capacity

The maximum number of key-value pairs that can be contained by the new dictionary. The dictionary starts with the same number of key-value pairs as *theDict* and can grow to this number of values (and it can have less).

Pass `0` to specify that the maximum capacity is not limited. If non-`0`, *capacity* must be greater than or equal to the count of *theDict*.

theDict

The dictionary to copy. The keys and values from the dictionary are copied as pointers into the new dictionary, not that which the values point to (if anything). The keys and values are also retained by the new dictionary. The count of the new dictionary is the same as the count of *theDict*. The new dictionary uses the same callbacks as *theDict*.

Return Value
A new dictionary that contains the same values as *theDict*. Ownership follows the Create Rule in *Memory Management Programming Guide for Core Foundation*.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Dumper
HID Utilities

MoreSCF

Declared In
CFDictionary.h

CFDictionaryRemoveAllValues

Removes all the key-value pairs from a dictionary, making it empty.

```
void CFDictionaryRemoveAllValues (
   CFMutableDictionaryRef theDict
);
```

Parameters

theDict
The dictionary to modify.

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

Related Sample Code
DockBrowser
SeeMyFriends

Declared In
CFDictionary.h

CFDictionaryRemoveValue

Removes a key-value pair.

```
void CFDictionaryRemoveValue (
   CFMutableDictionaryRef theDict,
   const void *key
);
```

Parameters

theDict
The dictionary to modify.

key
The key of the value to remove from *theDict*. If a key which matches *key* is present in *theDict*, the key-value pair is removed from the dictionary, otherwise this function does nothing ("remove if present").

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

Related Sample Code
CFPrefTopScores
DockBrowser
ImageClient
MoreSCF
SeeMyFriends

Declared In
CFDictionary.h

CFDictionaryReplaceValue

Replaces a value corresponding to a given key.

```
void CFDictionaryReplaceValue (
   CFMutableDictionaryRef theDict,
   const void *key,
   const void *value
);
```

Parameters

theDict

The dictionary to modify.

key

The key of the value to replace in *theDict*. If a key which matches *key* is present in the dictionary, the value is changed to the *value*, otherwise this function does nothing ("replace if present").

value

The new value for *key*. The *value* object is retained by *theDict* using the retain callback provided when *theDict* was created, and the old value is released. *value* must be of the type expected by the retain and release callbacks.

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

Related Sample Code
SeeMyFriends

Declared In
CFDictionary.h

CFDictionarySetValue

Sets the value corresponding to a given key.

```
void CFDictionarySetValue (
   CFMutableDictionaryRef theDict,
   const void *key,
   const void *value
);
```

Parameters

theDict

The dictionary to modify. If this parameter is a fixed-capacity dictionary and it is full before this operation, and the key does not exist in the dictionary, the behavior is undefined.

key

The key of the value to set in *theDict*. If a key which matches *key* is already present in the dictionary, only the value for the key is changed ("add if absent, replace if present"). If no key matches *key*, the key-value pair is added to the dictionary.

If a key-value pair is added, both *key* and *value* are retained by the dictionary, using the retain callback provided when *theDict* was created. *key* must be of the type expected by the key retain callback.

value

The value to add to or replace in *theDict*. *value* is retained using the value retain callback provided when *theDict* was created, and the previous value if any is released. *value* must be of the type expected by the retain and release callbacks.

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

Related Sample Code
Dictionary
From A View to A Movie
ImageClient
MoreSCF
STUCOtherDeviceTool

Declared In
`CFDictionary.h`

Data Types

CFMutableDictionaryRef

A reference to a mutable dictionary object.

```
typedef struct __CFDictionary *CFMutableDictionaryRef;
```

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

Declared In
`CFDictionary.h`

CFMutableSet Reference

Derived From:	*CFSet Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFSet.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFMutableSet manages dynamic sets. The basic interface for managing sets is provided by *CFSet Reference*. CFMutableSet adds functions to modify the contents of a set.

You create a mutable set object using either the `CFSetCreateMutable` (page 336) or `CFSetCreateMutableCopy` (page 337) function.

CFMutableSet provides several functions for adding and removing values from a set. The `CFSetAddValue` (page 335) function adds a value to a set and `CFSetRemoveValue` (page 338) removes a value from a set.

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

Functions

CFSetAddValue

Adds a value to a CFMutableSet object.

```
void CFSetAddValue (
   CFMutableSetRef theSet,
   const void *value
);
```

Parameters

theSet

The set to modify.

value

A CFType object or a pointer value to add to *theSet* (or the value itself, if it fits into the size of a pointer).

value is retained by *theSet* using the retain callback provided when *theSet* was created. If *value* is not of the type expected by the retain callback, the behavior is undefined. If *value* already exists in the collection, this function returns without doing anything.

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

Related Sample Code
CFLocalServer

Declared In
CFSet.h

CFSetCreateMutable

Creates an empty CFMutableSet object.

```
CFMutableSetRef CFSetCreateMutable (
   CFAllocatorRef allocator,
   CFIndex capacity,
   const CFSetCallBacks *callBacks
);
```

Parameters

allocator

The allocator to use to allocate memory for the new set and its storage for values. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

capacity

The maximum number of values that can be contained by the new set. The set starts empty and can grow to this number of values (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. The value must not be negative.

callBacks

A pointer to a CFSetCallBacks (page 512) structure initialized with the callbacks to use to retain, release, describe, and compare values in the set. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations. This parameter may be NULL, which is treated as if a valid structure of version 0 with all fields NULL had been passed in.

If any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a CFSetCallBacks structure, the behavior is undefined. If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

If the collection contains CFType objects only, then pass kCFTypeSetCallBacks (page 514) as this parameter to use the default callback functions.

Return Value
A new mutable set, or NULL if there was a problem creating the object. Ownership follows the Create Rule in *Memory Management Programming Guide for Core Foundation*.

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

Related Sample Code
CFLocalServer

Declared In
CFSet.h

CFSetCreateMutableCopy

Creates a new mutable set with the values from another set.

```
CFMutableSetRef CFSetCreateMutableCopy (
   CFAllocatorRef allocator,
   CFIndex capacity,
   CFSetRef theSet
);
```

Parameters

allocator

The allocator to use to allocate memory for the new set and its storage for values. Pass NULL or kCFAllocatorDefault (page 35) to use the current default allocator.

capacity

The maximum number of values that can be contained by the new set. The set starts with the same number of values as *theSet* and can grow to this number of values (and it can have less).

Pass 0 to specify that the maximum capacity is not limited. If non-0, *capacity* must be greater than or equal to the count of *theSet*.

theSet

The set to copy. The pointer values from *theSet* are copied into the new set. The values are also retained by the new set. The count of the new set is the same as the count of *theSet*. The new set uses the same callbacks as *theSet*.

Return Value
A new mutable set that contains the same values as *theSet*. Ownership follows the Create Rule in *Memory Management Programming Guide for Core Foundation*.

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

Declared In
CFSet.h

CFSetRemoveAllValues

Removes all values from a CFMutableSet object.

```
void CFSetRemoveAllValues (
   CFMutableSetRef theSet
);
```

Parameters

theSet

The set to modify.

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

Declared In
CFSet.h

CFSetRemoveValue

Removes a value from a CFMutableSet object.

```
void CFSetRemoveValue (
   CFMutableSetRef theSet,
   const void *value
);
```

Parameters

theSet

The set to modify.

value

The value to remove from *theSet*.

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

Related Sample Code
CFLocalServer

Declared In
CFSet.h

CFSetReplaceValue

Replaces a value in a CFMutableSet object.

```
void CFSetReplaceValue (
   CFMutableSetRef theSet,
   const void *value
);
```

Parameters

theSet

The set to modify.

value

The value to replace in *theSet*. If this value does not already exist in *theSet*, the function does nothing. You may pass the value itself instead of a pointer if it is pointer-size or less. The equal callback provided when *theSet* was created is used to compare. If the equal callback was NULL, pointer equality (in C, ==) is used. If *value*, or any other value in *theSet*, is not understood by the equal callback, the behavior is undefined.

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

Declared In
CFSet.h

CFSetSetValue

Sets a value in a CFMutableSet object.

```
void CFSetSetValue (
   CFMutableSetRef theSet,
   const void *value
);
```

Parameters

theSet

The set to modify.

value

The value to be set in *theSet*. If this value already exists in *theSet*, it is replaced. You may pass the value itself instead of a pointer to it if the value is pointer-size or less. If *theSet* is fixed-size and setting the value would increase its size beyond its capacity, the behavior is undefined.

Discussion
Depending on the implementation of the equal callback specified when creating *theSet*, the value that is replaced by *value* may not have the same pointer equality.

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

Declared In
CFSet.h

Data Types

CFMutableSetRef

A reference to a mutable set object.

```
typedef struct __CFSet *CFMutableSetRef;
```

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

Declared In
`CFSet.h`

CFMutableString Reference

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

Overview

CFMutableString manages dynamic strings. The basic interface for managing strings is provided by CFString. CFMutableString adds functions to modify the contents of a string.

CFMutableString is "toll-free bridged" with its Cocoa Foundation counterpart, `NSMutableString`. 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 `NSMutableString *` parameter, you can pass in a `CFMutableStringRef`, and in a function where you see a `CFMutableStringRef` parameter, you can pass in an NSMutableString instance. This also applies to concrete subclasses of NSMutableString. See Interchangeable Data Types for more information on toll-free bridging.

Functions

CFStringAppend

Appends the characters of a string to those of a CFMutableString object.

```
void CFStringAppend (
   CFMutableStringRef theString,
   CFStringRef appendedString
);
```

Parameters

theString

The string to which *appendedString* is appended. If *theString* is not a CFMutableString object, an assertion is raised.

appendedString

The string to append.

Discussion
This function reallocates the backing store of *theString* to accommodate the new length.

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

Related Sample Code
KillEveryOneButMe
MovieVideoChart
ProfileSystem
QTMetaData
String

Declared In
CFString.h

CFStringAppendCharacters

Appends a buffer of Unicode characters to the character contents of a CFMutableString object.

```
void CFStringAppendCharacters (
   CFMutableStringRef theString,
   const UniChar *chars,
   CFIndex numChars
);
```

Parameters

theString
The string to which the characters in *chars* are appended.

chars
A pointer to a buffer of Unicode characters.

numChars
The number of Unicode characters in *chars*.

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

Declared In
CFString.h

CFStringAppendCString

Appends a C string to the character contents of a CFMutableString object.

```
void CFStringAppendCString (
   CFMutableStringRef theString,
   const char *cStr,
   CFStringEncoding encoding
);
```

Parameters

theString

The string to which the characters from *cStr* are appended. If this value is not a CFMutableString object, an assertion is raised.

cStr

A pointer to a C string buffer.

encoding

The encoding of the characters in *cStr*.

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

Related Sample Code
audioburntest

simpleJavaLauncher

Declared In
CFString.h

CFStringAppendFormat

Appends a formatted string to the character contents of a CFMutableString object.

```
void CFStringAppendFormat (
   CFMutableStringRef theString,
   CFDictionaryRef formatOptions,
   CFStringRef format,
   ...
);
```

Parameters

theString

The string to which the formatted characters from *format* are appended. If this value is not a CFMutableString object, an assertion is raised.

formatOptions

A dictionary 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

A formatted string with `printf`-style specifiers.

...

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

Discussion
A formatted string is one with `printf`-style format specifiers embedded in the text such as `%d` (decimal), `%f` (double), and `%@` (Core Foundation object). The subsequent arguments, in order, are substituted for the specifiers in the character data appended to *`theString`*. You can also reorder the arguments in the string by using modifiers of the form "n$" with the format specifiers (for example, `%2$d`).

For more information on supported specifiers, see the relevant section in *String Programming Guide for Core Foundation*.

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

Related Sample Code
DockBrowser
DropDraw
KillEveryOneButMe
MovieVideoChart
String

Declared In
`CFString.h`

CFStringAppendFormatAndArguments

Appends a formatted string to the character contents of a CFMutableString object.

```
void CFStringAppendFormatAndArguments (
   CFMutableStringRef theString,
   CFDictionaryRef formatOptions,
   CFStringRef format,
   va_list arguments
);
```

Parameters

`theString`
The string to which the formatted characters from *`format`* are appended. If this value is not a CFMutableString object, an assertion is raised.

`formatOptions`
A dictionary 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`
A formatted string with `printf`-style specifiers.

`arguments`
List of values to be inserted in *`format`*.

Discussion
A formatted string is one with `printf`-style format specifiers embedded in the text such as `%d` (decimal), `%f` (double), and `%@` (Core Foundation object). The subsequent arguments, in order, are substituted for the specifiers in the character data appended to *`theString`*. You can also reorder the arguments in the string by using modifiers of the form "n$" with the format specifiers (for example, `%2$d`).

For more information on supported specifiers, see the relevant section in *String Programming Guide for Core Foundation*.

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

Declared In
`CFString.h`

CFStringAppendPascalString

Appends a Pascal string to the character contents of a CFMutableString object.

```
void CFStringAppendPascalString (
   CFMutableStringRef theString,
   ConstStr255Param pStr,
   CFStringEncoding encoding
);
```

Parameters

theString
The string to which the characters in *pStr* are appended. If this value is not a CFMutableString object, an assertion is raised.

pStr
A Pascal string buffer.

encoding
The string encoding of the characters in *pStr*.

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

Related Sample Code
String

Declared In
`CFString.h`

CFStringCapitalize

Changes the first character in each word of a string to uppercase (if it is a lowercase alphabetical character).

```
void CFStringCapitalize (
   CFMutableStringRef theString,
   CFLocaleRef locale
);
```

Parameters

theString
The string to be capitalized. If this value is not a CFMutableString object, an assertion is raised.

locale
A locale that specifies a particular language or region. Prior to Mac OS X v10.3, this parameter was an untyped pointer and not used.

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

Declared In
`CFString.h`

CFStringCreateMutable

Creates an empty CFMutableString object.

```
CFMutableStringRef CFStringCreateMutable (
   CFAllocatorRef alloc,
   CFIndex maxLength
);
```

Parameters

alloc

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

maxLength

The maximum number of Unicode characters that can be stored by the returned string. Pass `0` if there should be no character limit. Note that initially the string still has a length of `0`; this parameter simply specifies what the maximum size is. CFMutableString might try to optimize its internal storage by paying attention to this value.

Return Value
A new empty CFMutableString object or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates an empty (that is, content-less) CFMutableString object. You can add character data to this object with any of the `CFStringAppend...` functions, and thereafter you can insert, delete, replace, pad, and trim characters with the appropriate CFString functions. If the *maxLength* parameter is greater than `0`, any attempt to add characters beyond this limit results in a run-time error.

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

Related Sample Code
DockBrowser
KillEveryOneButMe
MovieVideoChart
QTMetaData
simpleJavaLauncher

Declared In
`CFString.h`

CFStringCreateMutableCopy

Creates a mutable copy of a string.

```
CFMutableStringRef CFStringCreateMutableCopy (
   CFAllocatorRef alloc,
   CFIndex maxLength,
   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.

maxLength

The maximum number of Unicode characters that can be stored by the returned object. Pass `0` if there should be no character limit. Note that initially the returned object still has the same length as the string argument; this parameter simply specifies what the maximum size is. CFString might try to optimize its internal storage by paying attention to this value.

theString

A string to copy.

Return Value

A string that has the same contents as *theString*. Returns `NULL` if there was a problem copying the object. Ownership follows the Create Rule.

Discussion

The returned mutable string is identical to the original string except for (perhaps) the mutability attribute. You can add character data to the returned string with any of the `CFStringAppend...` functions, and you can insert, delete, replace, pad, and trim characters with the appropriate CFString functions. If the *maxLength* parameter is greater than `0`, any attempt to add characters beyond this limit results in a run-time error.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CoreTextTest

DiagnosticAUs

MFSLives

MoreSCF

ProfileSystem

Declared In

`CFString.h`

CFStringCreateMutableWithExternalCharactersNoCopy

Creates a CFMutableString object whose Unicode character buffer is controlled externally.

```
CFMutableStringRef CFStringCreateMutableWithExternalCharactersNoCopy (
   CFAllocatorRef alloc,
   UniChar *chars,
   CFIndex numChars,
   CFIndex capacity,
   CFAllocatorRef externalCharactersAllocator
);
```

Parameters

alloc

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

chars

The Unicode character buffer for the new CFMutableString. Before calling, create this buffer on the stack or heap and optionally initialize it with Unicode character data. Upon return, the created CFString object keeps its own copy of the pointer to this buffer. You may pass in `NULL` if there is no initial buffer being provided.

numChars

The number of characters initially in the Unicode buffer pointed to by *chars*.

capacity

The capacity of the external buffer (*chars*); that is, the maximum number of Unicode characters that can be stored. This value should be `0` if no initial buffer is provided.

externalCharactersAllocator

The allocator to use to reallocate the external buffer when editing takes place and for deallocating the buffer when string is deallocated. If the default allocator is suitable for these purposes, pass `NULL`. If you do not want the new string to reallocate or deallocate memory for the buffer (that is, you assume responsibility for these things yourself), pass `kCFAllocatorNull`.

Return Value

A new mutable string, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

This function permits you to create a CFMutableString object whose backing store is an external Unicode character buffer—that is, a buffer that you control (or can control) entirely. This function allows you to take advantage of the features of CFString, particularly the CFMutableString functions that add and modify character data. But at the same time you can directly add, delete, modify, and examine the characters in the buffer. You can even replace the buffer entirely. If, however, you directly modify or replace the character buffer, you should inform the CFString object of this change with the `CFStringSetExternalCharactersNoCopy` (page 355) function.

If you mutate the character contents with the CFString functions, and the buffer needs to be enlarged, the CFString object calls the allocation callbacks specified for the allocator *externalCharactersAllocator*.

This function should be used in special circumstances where you want to create a CFString wrapper around an existing, potentially large `UniChar` buffer you own. Using this function causes the CFString object to forgo some of its internal optimizations, so it should be avoided in general use. That is, if you want to create a CFString object from a small `UniChar` buffer, and you don't need to continue owning the buffer, use one of the other creation functions (for instance `CFStringCreateWithCharacters` (page 555)) instead.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
String

Declared In
`CFString.h`

CFStringDelete

Deletes a range of characters in a string.

```
void CFStringDelete (
   CFMutableStringRef theString,
   CFRange range
);
```

Parameters

theString
A string from which characters are to be deleted.

range
The range of characters in *theString* to delete.

Discussion
The characters after the deleted range are adjusted to "fill in" the gap.

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

Related Sample Code
DockBrowser
HID Calibrator
HID Config Save
HID Explorer

Declared In
`CFString.h`

CFStringFindAndReplace

Replaces all occurrences of a substring within a given range.

```
CFIndex CFStringFindAndReplace (
   CFMutableStringRef theString,
   CFStringRef stringToFind,
   CFStringRef replacementString,
   CFRange rangeToSearch,
   CFOptionFlags compareOptions
);
```

Parameters

theString
The string to modify.

stringToFind
The substring to search for in `theString`.

replacementString
The replacement string for *stringToFind*.

rangeToSearch
The range within which to search 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 `CFStringCompareFlags` (page 589) for the available flags.

Return Value
The number of instances of *stringToFind* that were replaced.

Discussion
The possible values of *compareOptions* are combinations of the `kCFCompareCaseInsensitive` (page 591), `kCFCompareBackwards` (page 591), `kCFCompareNonliteral` (page 592), and `kCFCompareAnchored` (page 592) constants.

The `kCFCompareBackwards` option can be used to replace a substring starting from the end, which could produce different results. For example, if the parameter *theString* is "AAAAA", *stringToFind* is "AA", and *replacementString* is "B", then the result is normally "BBA". However, if the `kCFCompareBackwards` constant is used, the result is "ABB."

The `kCFCompareAnchored` option assures that only anchored but multiple instances are found (the instances must be consecutive at start or end). For example, if the parameter *theString* is "AAXAA", *stringToFind* is "A", and *replacementString* is "B", then the result is normally "BBXBB." However, if the `kCFCompareAnchored` constant is used, the result is "BBXAA."

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

Related Sample Code
simpleJavaLauncher

Declared In
`CFString.h`

CFStringFold

Folds a given string into the form specified by optional flags.

```
void CFStringFold (
    CFMutableStringRef theString,
    CFOptionFlags theFlags,
    CFLocaleRef theLocale
);
```

Parameters
theString
The string which is to be folded. If this parameter is not a valid mutable CFString, the behavior is undefined.

theFlags

The equivalency flags which describes the character folding form. See “String Comparison Flags” in *CFString Reference* for possible values. Only those flags containing the word "insensitive" are recognized; other flags are ignored.

Folding with `kCFCompareCaseInsensitive` removes case distinctions in accordance with the mapping specified by ftp://ftp.unicode.org/Public/UNIDATA/CaseFolding.txt. Folding with `kCFCompareDiacriticInsensitive` removes distinctions of accents and other diacritics. Folding with `kCFCompareWidthInsensitive` removes character width distinctions by mapping characters in the range `U+FF00-U+FFEF` to their ordinary equivalents.

theLocale

The locale to use for the operation. `NULL` specifies the canonical locale (the return value from `CFLocaleGetSystem` (page 250)).

The locale argument affects the case mapping 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.

Discussion

Character foldings are operations that convert any of a set of characters sharing similar semantics into a single representative from that set.

You can use this function to preprocess strings that are to be compared, searched, or indexed. Note that folding does not include normalization, so you must use `CFStringNormalize` (page 352) in addition to CFStringFold in order to obtain the effect of `kCFCompareNonliteral`.

Availability

Available in Mac OS X v10.5 and later.

Related Sample Code

DerivedProperty

Declared In

`CFString.h`

CFStringInsert

Inserts a string at a specified location in the character buffer of a CFMutableString object.

```
void CFStringInsert (
   CFMutableStringRef str,
   CFIndex idx,
   CFStringRef insertedStr
);
```

Parameters

str

The string to be modified. If this value is not a CFMutableString object, an assertion is raised.

index

The index of the character in *str* after which the new characters are to be inserted. If the index is out of bounds, an assertion is raised.

insertedStr

The string to insert into *str*.

Discussion
To accommodate the new characters, this function moves any existing characters to the right of the inserted characters the appropriate number of positions.

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

Related Sample Code
HID Explorer

Declared In
`CFString.h`

CFStringLowercase

Changes all uppercase alphabetical characters in a CFMutableString to lowercase.

```
void CFStringLowercase (
   CFMutableStringRef theString,
   CFLocaleRef locale
);
```

Parameters

`theString`

The string to be lowercased. If this value is not a CFMutableString object, an assertion is raised.

`locale`

The locale to use when the lowercasing operation is performed. Prior to Mac OS X v10.3 this parameter was an untyped pointer and not used.

The locale argument affects the case mapping 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.

Special Considerations

The *`locale`* parameter type changed from `void *` to `CFLocaleRef` in Mac OS X v10.3.

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

Related Sample Code
MoreSCF

Declared In
`CFString.h`

CFStringNormalize

Normalizes the string into the specified form as described in Unicode Technical Report #15.

```
void CFStringNormalize (
   CFMutableStringRef theString,
   CFStringNormalizationForm theForm
);
```

Parameters

theString

The string to be normalized.

theForm

The form to normalize *theString*.

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

Related Sample Code
DerivedProperty

MFSLives

MoreSCF

Declared In
CFString.h

CFStringPad

Enlarges a string, padding it with specified characters, or truncates the string.

```
void CFStringPad (
   CFMutableStringRef theString,
   CFStringRef padString,
   CFIndex length,
   CFIndex indexIntoPad
);
```

Parameters

theString

The string to modify.

padString

A string containing the characters with which to fill the extended character buffer. Pass `NULL` to truncate the string.

length

The new length of *theString*. If this length is greater than the current length, padding takes place; if it is less, truncation takes place.

indexIntoPad

The index of the character in *padString* with which to begin padding. If you are truncating the string represented by the object, this parameter is ignored.

Discussion
This function has two purposes. It either enlarges the character buffer of a CFMutableString object to a given length, padding the added length with a given character or characters, or it truncates the character buffer to a smaller size. The key parameter for this behavior is *length*; if it is greater than the current length of the represented string, padding takes place, and if it less than the current length, truncation occurs.

For example, say you have a string, `aMutStr`, containing the characters "abcdef". The call

```
CFStringPad(aMutStr, CFSTR("123"), 9, 1);
```

results in `aMutStr` containing "abcdef231". However, the following call

```
CFStringPad(aMutStr, NULL, 3, 0);
```

results in `aMutStr` containing "abc".

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

Declared In
`CFString.h`

CFStringReplace

Replaces part of the character contents of a CFMutableString object with another string.

```
void CFStringReplace (
   CFMutableStringRef theString,
   CFRange range,
   CFStringRef replacement
);
```

Parameters

`theString`

The string to modify. The characters are adjusted left or right (depending on the length of the substituted string) and the character buffer of the object is resized accordingly. If this value is not a CFMutableString object, an assertion is raised.

`range`

The range of characters in *`theString`* to replace.

`replacement`

The replacement string to put into *`theString`*.

Discussion
Although you can use this function to replace all characters in a CFMutableString object (by specifying a range of `(0, CFStringGetLength(theString))`), it is more convenient to use the `CFStringReplaceAll` (page 355) function for that purpose.

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

Related Sample Code
CIVideoDemoGL
MoreSCF
SpellingChecker CarbonCocoa Bundled
SpellingChecker-CarbonCocoa
SpellingChecker-CocoaCarbon

Declared In
`CFString.h`

CFStringReplaceAll

Replaces all characters of a CFMutableString object with other characters.

```
void CFStringReplaceAll (
   CFMutableStringRef theString,
   CFStringRef replacement
);
```

Parameters

theString

The string to modify. If this value is not a CFMutableString object, an assertion is raised.

replacement

The replacement string to put into *theString*.

Discussion

The character buffer of *theString* is resized according to the length of the new characters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFString.h

CFStringSetExternalCharactersNoCopy

Notifies a CFMutableString object that its external backing store of Unicode characters has changed.

```
void CFStringSetExternalCharactersNoCopy (
   CFMutableStringRef theString,
   UniChar *chars,
   CFIndex length,
   CFIndex capacity
);
```

Parameters

theString

The string to act as a "wrapper" for the external backing store (*chars*). If this value is not a CFMutableString object, an assertion is raised.

chars

The external (client-owned) Unicode buffer acting as the backing store for *theString*.

length

The current length of the contents of *chars* (in Unicode characters).

capacity

The capacity of the Unicode buffer—that is, the total number of Unicode characters that can be stored in it before the buffer has to be grown.

Discussion

You use this function to reallocate memory for a string, if necessary, and change its references to the data in the buffer. The object must have been created with the CFStringCreateMutableWithExternalCharactersNoCopy (page 347) function; see the discussion of this function for more information.

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

Declared In
`CFString.h`

CFStringTransform

Perform in-place transliteration on a mutable string.

```
Boolean CFStringTransform (
   CFMutableStringRef string,
   CFRange *range,
   CFStringRef transform,
   Boolean reverse
);
```

Parameters

string
The string to transform.

range
A pointer to the range over which the transformation is applied. `NULL` causes the whole string to be transformed. On return, *range* is modified to reflect the new range corresponding to the original range.

transform
A CFString object that identifies the transformation to apply. For a list of valid values, see "Transform Identifiers for CFStringTransform" (page 359). On Mac OS X v10.4 and later, you can also use any valid ICU transform ID defined in the ICU User Guide for Transforms.

reverse
A Boolean that, if `true`, specifies that the inverse transform should be used (if it exists).

Return Value
`true` if the transform is successful; otherwise `false`.

Discussion
The transformation represented by *transform* is applied to the given range of *string*, modifying it in place. Only the specified range is modified, but the transform may look at portions of the string outside that range for context. Reasons that the transform may be unsuccessful include an invalid transform identifier, and attempting to reverse an irreversible transform.

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

Declared In
`CFString.h`

CFStringTrim

Trims a specified substring from the beginning and end of a CFMutableString object.

```
void CFStringTrim (
   CFMutableStringRef theString,
   CFStringRef trimString
);
```

Parameters

theString

The string to trim. If this value is not a CFMutableString object, an assertion is raised.

trimString

The string to trim from *theString*. The characters of the trim string are treated as a substring and not individually; for example, if the mutable characters are "abc X" and the trim string is "XY", the mutable characters are not affected.

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

Related Sample Code
MoreSCF

Declared In
CFString.h

CFStringTrimWhitespace

Trims whitespace from the beginning and end of a CFMutableString object.

```
void CFStringTrimWhitespace (
   CFMutableStringRef theString
);
```

Parameters

theString

The string to trim. If this value is not a CFMutableString object, an assertion is raised.

Discussion
Whitespace for this function includes space characters, tabs, newlines, carriage returns, and any similar characters that do not have a visible representation.

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

Declared In
CFString.h

CFStringUppercase

Changes all lowercase alphabetical characters in a CFMutableString object to uppercase.

```
void CFStringUppercase (
   CFMutableStringRef theString,
   CFLocaleRef locale
);
```

Parameters

theString

The string to uppercase. If this value is not a CFMutableString object, an assertion is raised.

locale

A CFLocale object that specifies a particular language or region. Prior to Mac OS X v10.3, this parameter was an untyped pointer and not used.

The locale argument affects the case mapping 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.

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

Declared In
CFString.h

Data Types

CFMutableStringRef

A reference to a CFMutableString object.

```
typedef CFStringRef CFMutableStringRef;
```

Discussion
The type refers to a CFMutableString object, which “encapsulates” a Unicode string along with its length; the object has the attribute of being mutable, which means that its character contents can be modified. CFString is an opaque type that defines the characteristics and behavior of CFString objects, both immutable and mutable.

CFMutableString derives from CFString. Therefore, you can pass CFMutableString objects into functions accepting CFString objects.

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

Declared In
CFBase.h

Constants

String Normalization Forms

Unicode normalization forms as described in Unicode Technical Report #15.

```
enum CFStringNormalizationForm {
    kCFStringNormalizationFormD = 0,
    kCFStringNormalizationFormKD = 1,
    kCFStringNormalizationFormC = 2,
    kCFStringNormalizationFormKC = 3
};
typedef enum CFStringNormalizationForm CFStringNormalizationForm;
```

Constants

`kCFStringNormalizationFormD`
Canonical decomposition.
Available in Mac OS X v10.2 and later.
Declared in `CFString.h`.

`kCFStringNormalizationFormKD`
Compatibility decomposition.
Available in Mac OS X v10.2 and later.
Declared in `CFString.h`.

`kCFStringNormalizationFormC`
Canonical decomposition followed by canonical composition.
Available in Mac OS X v10.2 and later.
Declared in `CFString.h`.

`kCFStringNormalizationFormKC`
Compatibility decomposition followed by canonical composition.
Available in Mac OS X v10.2 and later.
Declared in `CFString.h`.

Transform Identifiers for CFStringTransform

Constants that identify transforms used with `CFStringTransform` (page 356).

```
const CFStringRef kCFStringTransformStripCombiningMarks;
const CFStringRef kCFStringTransformToLatin;
const CFStringRef kCFStringTransformFullwidthHalfwidth;
const CFStringRef kCFStringTransformLatinKatakana;
const CFStringRef kCFStringTransformLatinHiragana;
const CFStringRef kCFStringTransformHiraganaKatakana;
const CFStringRef kCFStringTransformMandarinLatin;
const CFStringRef kCFStringTransformLatinHangul;
const CFStringRef kCFStringTransformLatinArabic;
const CFStringRef kCFStringTransformLatinHebrew;
const CFStringRef kCFStringTransformLatinThai;
const CFStringRef kCFStringTransformLatinCyrillic;
const CFStringRef kCFStringTransformLatinGreek;
const CFStringRef kCFStringTransformToXMLHex;
const CFStringRef kCFStringTransformToUnicodeName;
const CFStringRef kCFStringTransformStripDiacritics;
```

Constants

`kCFStringTransformStripCombiningMarks`
The identifier of a transform to strip combining marks (accents or diacritics).

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformToLatin`
The identifier of a transform to transliterate all text possible to Latin script. Ideographs are transliterated as Mandarin Chinese.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformFullwidthHalfwidth`
The identifier of a reversible transform to convert full-width characters to their half-width equivalents.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinKatakana`
The identifier of a reversible transform to transliterate text to Katakana from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinHiragana`
The identifier of a reversible transform to transliterate text to Hiragana from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformHiraganaKatakana`
The identifier of a reversible transform to transliterate text to Katakana from Hiragana.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformMandarinLatin`
The identifier of a reversible transform to transliterate text to Latin from ideographs interpreted as Mandarin Chinese.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinHangul`
The identifier of a reversible transform to transliterate text to Hangul from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinArabic`
The identifier of a reversible transform to transliterate text to Arabic from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinHebrew`
The identifier of a reversible transform to transliterate text to Hebrew from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinThai`
The identifier of a reversible transform to transliterate text to Thai from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinCyrillic`
The identifier of a reversible transform to transliterate text to Cyrillic from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformLatinGreek`
The identifier of a reversible transform to transliterate text to Greek from Latin.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformToXMLHex`
The identifier of a reversible transform to transliterate characters other than printable ASCII to XML/HTML numeric entities.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformToUnicodeName`
The identifier of a reversible transform to transliterate characters other than printable ASCII (minus braces) to their Unicode character name in braces.

Examples include {AIRPLANE} and {GREEK CAPITAL LETTER PSI}.

Available in Mac OS X v10.4 and later.

Declared in `CFString.h`.

`kCFStringTransformStripDiacritics`
The identifier of a transform to remove diacritic markings.

Available in Mac OS X v10.5 and later.

Declared in `CFString.h`.

Discussion
On Mac OS X v10.4 and later, with `CFStringTransform` (page 356) you can also use any valid ICU transform ID defined in the ICU User Guide for Transforms.

CFNotificationCenter Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFNotificationCenter.h
Companion guide	Notification Programming Topics

Overview

A CFNotificationCenter object provides the means by which you can send a message, or notification, to any number of recipients, or observers, without having to know anything about the recipients. A notification message consists of a notification name (a CFString), a pointer value that identifies the object posting the notification, and an optional dictionary that contains additional information about the particular notification.

To register as an observer of a notification, you call `CFNotificationCenterAddObserver` (page 364), providing an identifier for your observer, the callback function that should be called when the notification is posted, and the name of the notification and the object in which you are interested. The observer identifier is passed back to the callback function, along with the notification information. You can use the identifier to distinguish multiple observers using the same callback function. The identifier is also used to unregister the observer with `CFNotificationCenterRemoveObserver` (page 370) and `CFNotificationCenterRemoveEveryObserver` (page 369).

To send a notification, you call `CFNotificationCenterPostNotification` (page 367), passing in the notification information. The notification center then looks up all the observers that registered for this notification and sends the notification information to their callback functions.

There are three types of CFNotificationCenter—a distributed notification center, a local notification center, and a Darwin notification center—an application may have at most one of each type. The distributed notification is obtained with `CFNotificationCenterGetDistributedCenter` (page 366). A distributed notification center delivers notifications between applications. In this case, the notification object must always be a CFString object and the notification dictionary must contain only property list values. The local and Darwin notification centers are available in Mac OS X version 10.4 and later, and obtained using `CFNotificationCenterGetLocalCenter` (page 367) and `CFNotificationCenterGetDarwinNotifyCenter` (page 366) respectively.

Unlike some other Core Foundation opaque types with names similar to a Cocoa Foundation class (such as CFString and `NSString`), CFNotificationCenter objects cannot be cast (“toll-free bridged”) to `NSNotificationCenter` objects or vice-versa.

Functions by Task

Accessing a Notification Center

`CFNotificationCenterGetDarwinNotifyCenter` (page 366)
Returns the application's Darwin notification center.

`CFNotificationCenterGetDistributedCenter` (page 366)
Returns the application's distributed notification center.

`CFNotificationCenterGetLocalCenter` (page 367)
Returns the application's local notification center.

Posting a Notification

`CFNotificationCenterPostNotification` (page 367)
Posts a notification for an object.

`CFNotificationCenterPostNotificationWithOptions` (page 368)
Posts a notification for an object using specified options.

Adding and Removing Observers

`CFNotificationCenterAddObserver` (page 364)
Registers an observer to receive notifications.

`CFNotificationCenterRemoveEveryObserver` (page 369)
Stops an observer from receiving any notifications from any object.

`CFNotificationCenterRemoveObserver` (page 370)
Stops an observer from receiving certain notifications.

Getting the CFNotificationCenter Type ID

`CFNotificationCenterGetTypeID` (page 367)
Returns the type identifier for the CFNotificationCenter opaque type.

Functions

CFNotificationCenterAddObserver

Registers an observer to receive notifications.

```
void CFNotificationCenterAddObserver (
   CFNotificationCenterRef center,
   const void *observer,
   CFNotificationCallback callBack,
   CFStringRef name,
   const void *object,
   CFNotificationSuspensionBehavior suspensionBehavior
);
```

Parameters

center

The notification center to which to add the observer.

observer

The observer. In Mac OS X v10.3 and later, this parameter may be `NULL`.

callBack

The callback function to call when *object* posts the notification named *name*.

name

The name of the notification to observe. If `NULL`, *callback* is called for any notification posted by *object*.

If *center* is a Darwin notification center, this value must *not* be `NULL`.

object

The object to observe. For distributed notifications, *object* must be a CFString object. If `NULL`, *callback* is called when a notification named *name* is posted by any object.

If *center* is a Darwin notification center, this value is ignored.

suspensionBehavior

Flag indicating how notifications should be handled when the application is in the background. See "Notification Delivery Suspension Behavior" (page 372) for the list of available values.

If *center* is a Darwin notification center, this value is ignored.

Discussion

The first time an observer is registered with a distributed notification center, the notification center creates a connection with the system-wide notification server and places a listening port into the common modes of the current thread's run loop. When a notification is delivered, it is processed on this initial thread, even if the observer that is receiving the notification registered for the notification on a different thread.

Because loaded frameworks may potentially spawn threads and add their own observers before your code executes, you cannot know for certain which thread will receive distributed notifications. If you need to control which thread processes a notification, your callback function must be able to forward the notification to the proper thread. You can use a CFMessagePort object or a custom CFRunLoopSource object to send notifications to the correct thread's run loop.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NotificationObserver

VideoViewer

XcodeClientServer

Declared In

`CFNotificationCenter.h`

CFNotificationCenterGetDarwinNotifyCenter

Returns the application's Darwin notification center.

```
CFNotificationCenterRef CFNotificationCenterGetDarwinNotifyCenter (
   void
);
```

Return Value
The application's Darwin notification center.

Discussion
This notification center is used to cover the `<notify.h>` Core OS notification mechanism (see `/usr/include/notify.h`). An application has only one Darwin notification center, so this function returns the same value each time it is called.

The Darwin Notify Center has no notion of per-user sessions, all notifications are system-wide. As with distributed notifications, the main thread's run loop must be running in one of the common modes (usually `kCFRunLoopDefaultMode`) for Darwin-style notifications to be delivered.

Important: Several function parameters are ignored by Darwin notification centers. To ensure future compatibility, you should pass `NULL` or `0` for all ignored arguments.

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

Declared In
`CFNotificationCenter.h`

CFNotificationCenterGetDistributedCenter

Returns the application's distributed notification center.

```
CFNotificationCenterRef CFNotificationCenterGetDistributedCenter (
   void
);
```

Return Value
The application's distributed notification center. An application has only one distributed notification center, so this function returns the same value each time it is called.

Discussion
A distributed notification center delivers notifications between applications. A notification object used with a distributed notification center must always be a CFString object and the notification dictionary must contain only property list values.

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

Related Sample Code
NotificationObserver
NotificationPoster
VideoViewer
XcodeClientServer

Declared In
CFNotificationCenter.h

CFNotificationCenterGetLocalCenter

Returns the application's local notification center.

```
CFNotificationCenterRef CFNotificationCenterGetLocalCenter (
   void
);
```

Return Value
The application's local notification center. An application has only one local notification center, so this function returns the same value each time it is called.

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

Declared In
CFNotificationCenter.h

CFNotificationCenterGetTypeID

Returns the type identifier for the CFNotificationCenter opaque type.

```
CFTypeID CFNotificationCenterGetTypeID (
   void
);
```

Return Value
The type identifier for the CFNotificationCenter opaque type.

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

Declared In
CFNotificationCenter.h

CFNotificationCenterPostNotification

Posts a notification for an object.

```
void CFNotificationCenterPostNotification (
   CFNotificationCenterRef center,
   CFStringRef name,
   const void *object,
   CFDictionaryRef userInfo,
   Boolean deliverImmediately
);
```

Parameters

center

The notification center to post the notification.

name

The name of the notification to post. This value must not be `NULL`.

object

The object posting the notification. If `NULL`, the notification is sent only to observers that are observing all objects. In other words, only observers that registered for the notification with a `NULL` value for *object* will receive the notification.

If you want to allow your clients to register for notifications using Cocoa APIs (see *NSNotificationCenter Class Reference*), then *object* must be a Core Foundation or Cocoa object.

For distributed notifications, *object* must be a CFString object.

If *center* is a Darwin notification center, this value is ignored.

userInfo

A dictionary passed to observers. You populate this dictionary with additional information describing the notification. For distributed notifications, the dictionary must contain only property list objects. This value may be `NULL`.

If *center* is a Darwin notification center, this value is ignored.

deliverImmediately

If `true`, the notification is delivered to all observers immediately, even if some observers are in suspended (background) applications and they requested different suspension behavior when registering for the notification. If `false`, each observer's requested suspension behavior is respected.

If *center* is a Darwin notification center, this value is ignored.

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

Related Sample Code
NotificationPoster
XcodeClientServer

Declared In
`CFNotificationCenter.h`

CFNotificationCenterPostNotificationWithOptions

Posts a notification for an object using specified options.

```
void CFNotificationCenterPostNotificationWithOptions (
   CFNotificationCenterRef center,
   CFStringRef name,
   const void *object,
   CFDictionaryRef userInfo,
   CFOptionFlags options
);
```

Parameters

center

The notification center to post the notification.

name

The name of the notification to post. This value must not be `NULL`.

object

The object posting the notification. If NULL, the notification is sent only to observers that are observing all objects. In other words, only observers that registered for the notification with a NULL value for *object* will receive the notification.

If you want to allow your clients to register for notifications using Cocoa APIs (see *NSNotificationCenter Class Reference*), then *object* must be a Core Foundation or Cocoa object.

For distributed notifications, *object* must be a CFString object.

If *center* is a Darwin notification center, this value is ignored.

userInfo

A dictionary to pass to observers. You populate this dictionary with additional information describing the notification. For distributed notifications, the dictionary must contain only property list objects. Can be NULL.

If *center* is a Darwin notification center, this value is ignored.

options

Specifies if the notification should be posted immediately, or to all sessions. See "Notification Posting Options" (page 373) for possible values.

If *center* is a Darwin notification center, this value is ignored.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFNotificationCenter.h

CFNotificationCenterRemoveEveryObserver

Stops an observer from receiving any notifications from any object.

```
void CFNotificationCenterRemoveEveryObserver (
   CFNotificationCenterRef center,
   const void *observer
);
```

Parameters

center

The notification center from which to remove observers.

observer

The observer. This value must not be NULL.

Discussion
If you no longer want an observer to receive any notifications, perhaps because the observer is being deallocated, you can call this function to unregister the observer from all the notifications for which it had previously registered.

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

Related Sample Code
NotificationObserver

XcodeClientServer

Declared In
`CFNotificationCenter.h`

CFNotificationCenterRemoveObserver

Stops an observer from receiving certain notifications.

```
void CFNotificationCenterRemoveObserver (
   CFNotificationCenterRef center,
   const void *observer,
   CFStringRef name,
   const void *object
);
```

Parameters

center
The notification center to modify.

observer
The observer. This value must not be `NULL`.

name
The name of the notification to stop observing. If `NULL`, *observer* stops receiving callbacks for all notifications posted by *object*.

object
The object to stop observing. For distributed notifications, *object* must be a CFString object. If `NULL`, *observer* stops receiving callbacks for all objects posting notifications named *name*.

If *center* is a Darwin notification center, this value is ignored.

Discussion
If both *name* and *object* are `NULL`, this function unregisters *observer* from all the notifications for which it had previously registered with *center*.

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

Related Sample Code
NotificationObserver
VideoViewer
XcodeClientServer

Declared In
`CFNotificationCenter.h`

Callbacks

CFNotificationCallback

Callback function invoked for each observer of a notification when the notification is posted.

```
typedef void (*CFNotificationCallback) (
    CFNotificationCenterRef center,
    void *observer,
    CFStringRef name,
    const void *object,
    CFDictionaryRef userInfo
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFNotificationCenterRef center,
    void *observer,
    CFStringRef name,
    const void *object,
    CFDictionaryRef userInfo
);
```

Parameters

center

The notification center handling the notification.

observer

An arbitrary value, other than `NULL`, that identifies the observer.

name

The name of the notification being posted.

object

An arbitrary value that identifies the object posting the notification. For distributed notifications, *object* is always a CFString object. This value could be `NULL`.

userInfo

A dictionary containing additional information regarding the notification. This value could be `NULL`.

If the notification center is a Darwin notification center, this value must be ignored.

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

Declared In
`CFNotificationCenter.h`

Data Types

CFNotificationCenterRef

The type of a reference to a CFNotificationCenter.

```
typedef struct *CFNotificationCenterRef;
```

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

Declared In
CFNotificationCenter.h

Constants

Notification Delivery Suspension Behavior

Suspension flags that indicate how distributed notifications should be handled when the receiving application is in the background.

```
enum CFNotificationSuspensionBehavior {
    CFNotificationSuspensionBehaviorDrop = 1,
    CFNotificationSuspensionBehaviorCoalesce = 2,
    CFNotificationSuspensionBehaviorHold = 3,
    CFNotificationSuspensionBehaviorDeliverImmediately = 4
};
typedef enum CFNotificationSuspensionBehavior CFNotificationSuspensionBehavior;
```

Constants

`CFNotificationSuspensionBehaviorDrop`
The server will not queue any notifications of the specified name and object while the receiving application is in the background.

Available in Mac OS X v10.0 and later.

Declared in `CFNotificationCenter.h`.

`CFNotificationSuspensionBehaviorCoalesce`
The server will only queue the last notification of the specified name and object; earlier notifications are dropped.

Available in Mac OS X v10.0 and later.

Declared in `CFNotificationCenter.h`.

`CFNotificationSuspensionBehaviorHold`
The server will hold all matching notifications until the queue has been filled (queue size determined by the server) at which point the server may flush queued notifications.

Available in Mac OS X v10.0 and later.

Declared in `CFNotificationCenter.h`.

`CFNotificationSuspensionBehaviorDeliverImmediately`
The server will deliver notifications of the specified name and object whether or not the application is in the background. When a notification with this suspension behavior is matched, it has the effect of first flushing any queued notifications.

Available in Mac OS X v10.0 and later.

Declared in `CFNotificationCenter.h`.

Discussion
An application selects the suspension behavior for a given notification when it registers an observer for that notification with `CFNotificationCenterAddObserver` (page 364).

Notification Posting Options

Possible options when posting notifications.

```
enum {
    kCFNotificationDeliverImmediately = (1 << 0),
    kCFNotificationPostToAllSessions = (1 << 1)
};
```

Constants

`kCFNotificationDeliverImmediately`

Delivers the notification immediately.

Available in Mac OS X v10.3 and later.

Declared in `CFNotificationCenter.h`.

`kCFNotificationPostToAllSessions`

Delivers the notification to all sessions.

Available in Mac OS X v10.3 and later.

Declared in `CFNotificationCenter.h`.

Discussion

Use these constants when calling the `CFNotificationCenterPostNotificationWithOptions` (page 368) function.

CFNull Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBase.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

The CFNull opaque type defines a unique object used to represent null values in collection objects (which don't allow `NULL` values). CFNull objects are neither created nor destroyed. Instead, a single CFNull constant object—`kCFNull` (page 376)—is defined and is used wherever a null value is needed.

The CFNull opaque type is available in Mac OS X v10.2 and later.

Functions

CFNullGetTypeID

Returns the type identifier for the CFNull opaque type.

```
CFTypeID CFNullGetTypeID (
   void
);
```

Return Value
The type identifier for the CFNull opaque type.

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

Declared In
`CFBase.h`

Data Types

CFNullRef

A reference to a CFNull object.

```
typedef const struct __CFNull *CFNullRef;
```

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

Declared In
`CFBase.h`

Constants

Predefined Value

Predefined CFNull object.

```
const CFNullRef kCFNull;
```

Constants
`kCFNull`
The singleton CFNull object.
Available in Mac OS X v10.2 and later.
Declared in `CFBase.h`.

CFNumber Reference

Derived From:	CFPropertyList : CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFNumber.h
Companion guide	Property List Programming Topics for Core Foundation

Overview

CFNumber encapsulates C scalar (numeric) types. It provides functions for setting and accessing the value as any basic C type. It also provides a compare function to determine the ordering of two CFNumber objects. CFNumber objects are used to wrap numerical values for use in Core Foundation property lists and collections.

CFNumber objects are not intended as a replacement for C scalar values and should not be used in APIs or implementations where scalar values are more appropriate and efficient.

Note: In order to improve performance, some commonly-used numbers (such as `0` and `1`) are uniqued. You should not expect that allocating multiple CFNumber instances will necessarily result in distinct objects.

CFNumber is "toll-free bridged" with its Cocoa Foundation counterpart, NSNumber. 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 `NSNumber *` parameter, you can pass in a `CFNumberRef`, and in a function where you see a `CFNumberRef` parameter, you can pass in an NSNumber instance. This fact also applies to concrete subclasses of NSNumber. See Integrating Carbon and Cocoa in Your Application for more information on toll-free bridging.

Functions by Task

Creating a Number

`CFNumberCreate` (page 379)
Creates a CFNumber object using a specified value.

Getting Information About Numbers

`CFNumberGetByteSize` (page 380)
Returns the number of bytes used by a CFNumber object to store its value.

`CFNumberGetType` (page 380)
Returns the type used by a CFNumber object to store its value.

`CFNumberGetValue` (page 381)
Obtains the value of a CFNumber object cast to a specified type.

`CFNumberIsFloatType` (page 382)
Determines whether a CFNumber object contains a value stored as one of the defined floating point types.

Comparing Numbers

`CFNumberCompare` (page 378)
Compares two CFNumber objects and returns a comparison result.

Getting the CFNumber Type ID

`CFNumberGetTypeID` (page 381)
Returns the type identifier for the CFNumber opaque type.

Functions

CFNumberCompare

Compares two CFNumber objects and returns a comparison result.

```
CFComparisonResult CFNumberCompare (
   CFNumberRef number,
   CFNumberRef otherNumber,
   void *context
);
```

Parameters

number
The first CFNumber object to compare.

otherNumber
The second CFNumber object to compare.

context
Pass `NULL`.

Return Value
A `CFComparisonResult` (page 802) constant that indicates whether *number* is equal to, less than, or greater than *otherNumber*.

Discussion
When comparing two CFNumber objects using this function, one or both objects can represent a special-case number such as signed 0, signed infinity, or NaN.

The following rules apply:

- Negative 0 compares less than positive 0.
- Positive infinity compares greater than everything except itself, to which it compares equal.
- Negative infinity compares less than everything except itself, to which it compares equal.
- If both numbers are NaN, then they compare equal.
- If only one of the numbers is NaN, then the NaN compares greater than the other number if it is negative, and smaller than the other number if it is positive.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFNumber.h`

CFNumberCreate

Creates a CFNumber object using a specified value.

```
CFNumberRef CFNumberCreate (
   CFAllocatorRef allocator,
   CFNumberType theType,
   const void *valuePtr
);
```

Parameters

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

theType
A constant that specifies the data type of the value to convert. See Number Types (page 383) for a list of possible values.

valuePtr
A pointer to the value for the returned number object.

Return Value
A new number with the value specified by *valuePtr*. Ownership follows the Create Rule.

Discussion
The *theType* parameter is not necessarily preserved when creating a new CFNumber object. The CFNumber object will be created using whatever internal storage type the creation function deems appropriate. Use the function `CFNumberGetType` (page 380) to find out what type the CFNumber object used to store your value.

Availability
Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code
audioburntest
BetterAuthorizationSample
BSDLLCTest
HID Explorer
QTMetaData

Declared In
CFNumber.h

CFNumberGetByteSize

Returns the number of bytes used by a CFNumber object to store its value.

```
CFIndex CFNumberGetByteSize (
   CFNumberRef number
);
```

Parameters

number
The CFNumber object to examine.

Return Value
The size in bytes of the value contained in *number*.

Discussion
Because a CFNumber object might store a value using a type different from that of the original value with which it was created, this function may return a size different from the size of the original value's type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFNumber.h

CFNumberGetType

Returns the type used by a CFNumber object to store its value.

```
CFNumberType CFNumberGetType (
   CFNumberRef number
);
```

Parameters

number
The CFNumber object to examine.

Return Value
A constant that indicates the data type of the value contained in *number*. See Number Types (page 383) for a list of possible values.

Discussion
The type specified in the call to `CFNumberCreate` (page 379) is not necessarily preserved when a new CFNumber object is created—it uses whatever internal storage type the creation function deems appropriate.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFNumber.h`

CFNumberGetTypeID

Returns the type identifier for the CFNumber opaque type.

```
CFTypeID CFNumberGetTypeID (
   void
);
```

Return Value
The type identifier for the CFNumber opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BetterAuthorizationSample
BSDLLCTest
CFFTPSample
HID Manager Basics
MoreSCF

Declared In
`CFNumber.h`

CFNumberGetValue

Obtains the value of a CFNumber object cast to a specified type.

```
Boolean CFNumberGetValue (
   CFNumberRef number,
   CFNumberType theType,
   void *valuePtr
);
```

Parameters

number
The CFNumber object to examine.

theType
A constant that specifies the data type to return. See Number Types (page 383) for a list of possible values.

valuePtr
On return, contains the value of *number*.

Return Value
`true` if the operation was successful, otherwise `false`.

Discussion
If the argument type differs from the return type, and the conversion is lossy or the return value is out of range, then this function passes back an approximate value in *valuePtr* and returns `false`.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioCDSample
BSDLLCTest
GLUT
HID Manager Basics
MoreSCF

Declared In
`CFNumber.h`

CFNumberIsFloatType

Determines whether a CFNumber object contains a value stored as one of the defined floating point types.

```
Boolean CFNumberIsFloatType (
   CFNumberRef number
);
```

Parameters
number
The CFNumber object to examine.

Return Value
`true` if *number*'s value is one of the defined floating point types, otherwise `false`. The valid floating point types are listed in Number Types (page 383).

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFNumber.h`

Data Types

CFNumberRef

A reference to a CFNumber object.

```
typedef const struct __CFNumber *CFNumberRef;
```

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

Declared In
`CFNumber.h`

Constants

Number Types

Flags used by CFNumber to indicate the data type of a value.

```
enum CFNumberType {
    kCFNumberSInt8Type = 1,
    kCFNumberSInt16Type = 2,
    kCFNumberSInt32Type = 3,
    kCFNumberSInt64Type = 4,
    kCFNumberFloat32Type = 5,
    kCFNumberFloat64Type = 6,
    kCFNumberCharType = 7,
    kCFNumberShortType = 8,
    kCFNumberIntType = 9,
    kCFNumberLongType = 10,
    kCFNumberLongLongType = 11,
    kCFNumberFloatType = 12,
    kCFNumberDoubleType = 13,
    kCFNumberCFIndexType = 14,
    kCFNumberNSIntegerType = 15,
    kCFNumberCGFloatType = 16,
    kCFNumberMaxType = 16
};
typedef enum CFNumberType CFNumberType;
```

Constants
`kCFNumberSInt8Type`
Eight-bit, signed integer. The `SInt8` data type is defined in `MacTypes.h`.
Available in Mac OS X v10.0 and later.
Declared in `CFNumber.h`.

`kCFNumberSInt16Type`
Sixteen-bit, signed integer. The `SInt16` data type is defined in `MacTypes.h`.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberSInt32Type`
Thirty-two-bit, signed integer. The `SInt32` data type is defined in `MacTypes.h`.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberSInt64Type`
Sixty-four-bit, signed integer. The `SInt64` data type is defined in `MacTypes.h`.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberFloat32Type`
Thirty-two-bit real. The `Float32` data type is defined in `MacTypes.h`.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberFloat64Type`
Sixty-four-bit real. The `Float64` data type is defined in `MacTypes.h` and conforms to the 64-bit IEEE 754 standard.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberCharType`
Basic C `char` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberShortType`
Basic C `short` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberIntType`
Basic C `int` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberLongType`
Basic C `long` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberLongLongType`
Basic C `long long` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberFloatType`
Basic C `float` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberDoubleType`
Basic C `double` type.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberCFIndexType`
CFIndex value.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberNSIntegerType`
`NSInteger` value.

Available in Mac OS X v10.5 and later.

Declared in `CFNumber.h`.

`kCFNumberCGFloatType`
`CGFloat` value.

Available in Mac OS X v10.5 and later.

Declared in `CFNumber.h`.

`kCFNumberMaxType`
Same as `kCFNumberCGFloatType`.

Note that on Mac OS X v10.4, `kCFNumberMaxType` was the same as `kCFNumberCFIndexType`.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

Discussion
The type specified in the call to `CFNumberCreate` (page 379) is not necessarily preserved when creating a new CFNumber object. A CFNumber object uses whatever internal storage type the creation function deems appropriate. Use the `CFNumberGetType` (page 380) function to find out what type the CFNumber object used to store your value.

Predefined Values

CFNumber provides some predefined number values.

```
const CFNumberRef kCFNumberNaN;
const CFNumberRef kCFNumberNegativeInfinity;
const CFNumberRef kCFNumberPositiveInfinity;
```

Constants
`kCFNumberNaN`
"Not a Number." This value is often the result of an invalid operation, such as the square-root of a negative number.

Available in Mac OS X v10.0 and later.

Declared in `CFNumber.h`.

`kCFNumberNegativeInfinity`
Designates a negative infinity value.
Available in Mac OS X v10.0 and later.
Declared in `CFNumber.h`.

`kCFNumberPositiveInfinity`
Designates a positive infinity value.
Available in Mac OS X v10.0 and later.
Declared in `CFNumber.h`.

CFNumberFormatter Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFNumberFormatter.h
Companion guide	Data Formatting Guide for Core Foundation

Overview

CFNumberFormatter objects format the textual representations of *CFNumber* objects, and convert textual representations of numbers into *CFNumber* objects. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. You specify how strings are formatted and parsed by setting a format string and other properties of a `CFNumberFormatter` object. The format of the format string itself is defined by Unicode Technical Standard #35.

Note that `CFNumberFormatter` is not thread-safe. Do not use a single instance from multiple threads.

The `CFNumberFormatter` opaque type is available in Mac OS X v10.3 and later.

Unlike some other Core Foundation opaque types with names similar to a corresponding Cocoa Foundation class (such as `CFString` and NSString), `CFNumberFormatter` objects cannot be cast ("toll-free bridged") to `NSNumberFormatter` objects.

Functions by Task

Creating a Number Formatter

`CFNumberFormatterCreate` (page 389)
Creates a new CFNumberFormatter object, localized to the given locale, which will format numbers to the given style.

Configuring a Number Formatter

`CFNumberFormatterSetFormat` (page 394)
Sets the format string of a number formatter.

`CFNumberFormatterSetProperty` (page 395)
Sets a number formatter property using a key-value pair.

Formatting Values

`CFNumberFormatterCreateNumberFromString` (page 389)
Returns a number object representing a given string.

`CFNumberFormatterCreateStringWithNumber` (page 390)
Returns a string representation of the given number using the specified number formatter.

`CFNumberFormatterCreateStringWithValue` (page 391)
Returns a string representation of the given number or value using the specified number formatter.

`CFNumberFormatterGetDecimalInfoForCurrencyCode` (page 391)
Returns the number of fraction digits that should be displayed, and the rounding increment, for a given currency.

`CFNumberFormatterGetValueFromString` (page 394)
Returns a number or value representing a given string.

Examining a Number Formatter

`CFNumberFormatterCopyProperty` (page 388)
Returns a copy of a number formatter's value for a given key.

`CFNumberFormatterGetFormat` (page 392)
Returns a format string for the given number formatter object.

`CFNumberFormatterGetLocale` (page 393)
Returns the locale object used to create the given number formatter object.

`CFNumberFormatterGetStyle` (page 393)
Returns the number style used to create the given number formatter object.

Getting the CFNumberFormatter Type ID

`CFNumberFormatterGetTypeID` (page 393)
Returns the type identifier for the `CFNumberFormatter` opaque type.

Functions

CFNumberFormatterCopyProperty

Returns a copy of a number formatter's value for a given key.

```
CFTypeRef CFNumberFormatterCopyProperty (
   CFNumberFormatterRef formatter,
   CFStringRef key
);
```

Parameters

formatter
The number formatter to examine.

key

A property key. See "Number Formatter Property Keys" (page 398) for valid values.

Return Value
A `CFType` object that is a copy of the property value for *key*. Returns `NULL` if there is no value specified for *key*. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterCreate

Creates a new CFNumberFormatter object, localized to the given locale, which will format numbers to the given style.

```
CFNumberFormatterRef CFNumberFormatterCreate (
   CFAllocatorRef allocator,
   CFLocaleRef locale,
   CFNumberFormatterStyle style
);
```

Parameters

alloc

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

locale

A locale to use for localization. If `NULL`, the function uses the default system locale. Use `CFLocaleCopyCurrent` (page 242) to specify the locale of the current user.

style

A number style. See "Number Formatter Styles" (page 397) for possible values.

Return Value
A new number formatter, localized to the given locale, which will format numbers using the given style. Returns `NULL` if there was a problem creating the formatter. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterCreateNumberFromString

Returns a number object representing a given string.

```
CFNumberRef CFNumberFormatterCreateNumberFromString (
   CFAllocatorRef allocator,
   CFNumberFormatterRef formatter,
   CFStringRef string,
   CFRange *rangep,
   CFOptionFlags options
);
```

Parameters

alloc

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

formatter

The number formatter to use.

string

The string to parse.

rangep

A reference to a range that specifies the substring of *string* to be parsed. If `NULL`, the whole string is parsed. On return, contains the range of the actual extent of the parse (may be less than the given range).

options

Specifies various configuration options to change the behavior of the parse. Currently, `kCFNumberFormatterParseIntegersOnly` (page 403) is the only possible value for this parameter.

Return Value

A new number that represents the given string. Returns `NULL` if there was a problem creating the number. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFNumberFormatter.h`

CFNumberFormatterCreateStringWithNumber

Returns a string representation of the given number using the specified number formatter.

```
CFStringRef CFNumberFormatterCreateStringWithNumber (
   CFAllocatorRef allocator,
   CFNumberFormatterRef formatter,
   CFNumberRef number
);
```

Parameters

alloc

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

formatter

The number formatter to use.

number

The number from which to create a string representation.

Return Value
A new string that represents the given number in the specified format. Returns `NULL` if there was a problem creating the string. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterCreateStringWithValue

Returns a string representation of the given number or value using the specified number formatter.

```
CFStringRef CFNumberFormatterCreateStringWithValue (
   CFAllocatorRef allocator,
   CFNumberFormatterRef formatter,
   CFNumberType numberType,
   const void *valuePtr
);
```

Parameters

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

formatter
The number formatter to use.

numberType
The type of value that *valuePtr* references. Valid values are listed in `CFNumberType` (page 383).

valuePtr
A pointer to the value to be converted.

Return Value
A new string that represents the given number or value formatted by *formatter*. Returns `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterGetDecimalInfoForCurrencyCode

Returns the number of fraction digits that should be displayed, and the rounding increment, for a given currency.

```
Boolean CFNumberFormatterGetDecimalInfoForCurrencyCode (
   CFStringRef currencyCode,
   int32_t *defaultFractionDigits,
   double *roundingIncrement
);
```

Parameters

currencyCode

A string containing a ISO 4217 3-letter currency code. For example, AUD for Australian Dollars, EUR for Euros.

defaultFractionDigits

Upon return, contains the number of fraction digits that should be displayed for the currency specified by *currencyCode*.

roundingIncrement

Upon return, contains the rounding increment for the currency specified by *currencyCode*, or `0.0` if no rounding is done by the currency.

Return Value

`true` if the information was obtained successfully, otherwise `false` (for example, if the currency code is unknown or the information is not available).

Discussion

The returned values are not localized because these are properties of the currency.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFNumberFormatter.h`

CFNumberFormatterGetFormat

Returns a format string for the given number formatter object.

```
CFStringRef CFNumberFormatterGetFormat (
   CFNumberFormatterRef formatter
);
```

Parameters

formatter

The number formatter to examine.

Return Value

The format string for *formatter* as was specified by calling the `CFNumberFormatterSetFormat` (page 394) function, or derived from the number formatter's style. The format of this string is defined by Unicode Technical Standard #35.. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFNumberFormatter.h`

CFNumberFormatterGetLocale

Returns the locale object used to create the given number formatter object.

```
CFLocaleRef CFNumberFormatterGetLocale (
   CFNumberFormatterRef formatter
);
```

Parameters

formatter
The number formatter to examine.

Return Value

The locale used to create *formatter*. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

CFNumberFormatter.h

CFNumberFormatterGetStyle

Returns the number style used to create the given number formatter object.

```
CFNumberFormatterStyle CFNumberFormatterGetStyle (
   CFNumberFormatterRef formatter
);
```

Parameters

formatter
The number formatter to examine.

Return Value

The number style used to create *formatter*.

Availability

Available in Mac OS X v10.3 and later.

Declared In

CFNumberFormatter.h

CFNumberFormatterGetTypeID

Returns the type identifier for the `CFNumberFormatter` opaque type.

```
CFTypeID CFNumberFormatterGetTypeID (
   void
);
```

Return Value

The type identifier for the `CFNumberFormatter` opaque type.

Availability

Available in Mac OS X v10.3 and later.

Declared In
CFNumberFormatter.h

CFNumberFormatterGetValueFromString

Returns a number or value representing a given string.

```
Boolean CFNumberFormatterGetValueFromString (
   CFNumberFormatterRef formatter,
   CFStringRef string,
   CFRange *rangep,
   CFNumberType numberType,
   void *valuePtr
);
```

Parameters

formatter
The number formatter to use.

string
The string to parse.

rangep
A reference to a range that specifies the substring of *string* to be parsed. If NULL, the whole string is parsed. Upon return, contains the range of the actual extent of the parse (may be less than the given range).

numberType
The type of value that *valuePtr* references. Valid values are listed in CFNumberType (page 383).

valuePtr
Upon return, contains a number or value representing the string in the specified format. You are responsible for releasing this value.

Return Value
true if the string was parsed successfully, otherwise false.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFNumberFormatter.h

CFNumberFormatterSetFormat

Sets the format string of a number formatter.

```
void CFNumberFormatterSetFormat (
   CFNumberFormatterRef formatter,
   CFStringRef formatString
);
```

Parameters

formatter
The number formatter to modify.

formatString

The format string to be used by *formatter*. The format of this string is defined by Unicode Technical Standard #35.

Discussion
The format string may override other properties previously set using other functions. If this function is not called, the default value of the format string is derived from the number formatter's style.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFNumberFormatter.h

CFNumberFormatterSetProperty

Sets a number formatter property using a key-value pair.

```
void CFNumberFormatterSetProperty (
   CFNumberFormatterRef formatter,
   CFStringRef key,
   CFTypeRef value
);
```

Parameters

formatter

The number formatter to modify.

key

The name of the property of *formatter* to set. See "Number Formatter Property Keys" (page 398) for a description of possible values.

value

The value of the specified key. This must be an instance of the correct `CFType` object for the corresponding key.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CFNumberFormatter.h

Data Types

CFNumberFormatterOptionFlags

Type for constants specifying how numbers should be parsed.

```
typedef CFOptionFlags CFNumberFormatterOptionFlags;
```

Discussion
For values, see "Number Format Options" (page 403).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterPadPosition

Type for constants specifying how numbers should be padded.

```
typedef CFIndex CFNumberFormatterPadPosition;
```

Discussion
For values, see "Padding Positions" (page 404).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterRef

A reference to a `CFNumberFormatter` object.

```
typedef struct __CFNumberFormatter *CFNumberFormatterRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

CFNumberFormatterStyle

Type for constants specifying a formatter style.

```
typedef CFIndex CFNumberFormatterStyle;
```

Discussion
For values, see "Number Formatter Styles" (page 397).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFNumberFormatter.h`

Constants

Number Formatter Styles

Predefined number format styles.

```
enum {
    kCFNumberFormatterNoStyle = 0,
    kCFNumberFormatterDecimalStyle = 1,
    kCFNumberFormatterCurrencyStyle = 2,
    kCFNumberFormatterPercentStyle = 3,
    kCFNumberFormatterScientificStyle = 4,
    kCFNumberFormatterSpellOutStyle = 5
};
```

Constants

`kCFNumberFormatterNoStyle`
Specifies no style.
Available in Mac OS X v10.3 and later.
Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterDecimalStyle`
Specifies a decimal style format.
Available in Mac OS X v10.3 and later.
Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterCurrencyStyle`
Specifies a currency style format.
Available in Mac OS X v10.3 and later.
Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPercentStyle`
Specifies a percent style format.
Available in Mac OS X v10.3 and later.
Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterScientificStyle`
Specifies a scientific style format.
Available in Mac OS X v10.3 and later.
Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterSpellOutStyle`
Specifies a spelled out format.
Available in Mac OS X v10.4 and later.
Declared in `CFNumberFormatter.h`.

Discussion

The format for these number styles is not exact because they depend on the locale, user preference settings, and operating system version. Do not use these constants if you want an exact format (for example, if you are parsing data in a given format). In general, however, you are encouraged to use these styles to accommodate user preferences.

Declared In
CFNumberFormatter.h

Number Formatter Property Keys

The keys used in key-value pairs to specify the value of number formatter properties.

```
const CFStringRef kCFNumberFormatterCurrencyCode;
const CFStringRef kCFNumberFormatterDecimalSeparator;
const CFStringRef kCFNumberFormatterCurrencyDecimalSeparator;
const CFStringRef kCFNumberFormatterAlwaysShowDecimalSeparator;
const CFStringRef kCFNumberFormatterGroupingSeparator;
const CFStringRef kCFNumberFormatterUseGroupingSeparator;
const CFStringRef kCFNumberFormatterPercentSymbol;
const CFStringRef kCFNumberFormatterZeroSymbol;
const CFStringRef kCFNumberFormatterNaNSymbol;
const CFStringRef kCFNumberFormatterInfinitySymbol;
const CFStringRef kCFNumberFormatterMinusSign;
const CFStringRef kCFNumberFormatterPlusSign;
const CFStringRef kCFNumberFormatterCurrencySymbol;
const CFStringRef kCFNumberFormatterExponentSymbol;
const CFStringRef kCFNumberFormatterMinIntegerDigits;
const CFStringRef kCFNumberFormatterMaxIntegerDigits;
const CFStringRef kCFNumberFormatterMinFractionDigits;
const CFStringRef kCFNumberFormatterMaxFractionDigits;
const CFStringRef kCFNumberFormatterGroupingSize;
const CFStringRef kCFNumberFormatterSecondaryGroupingSize;
const CFStringRef kCFNumberFormatterRoundingMode;
const CFStringRef kCFNumberFormatterRoundingIncrement;
const CFStringRef kCFNumberFormatterFormatWidth;
const CFStringRef kCFNumberFormatterPaddingPosition;
const CFStringRef kCFNumberFormatterPaddingCharacter;
const CFStringRef kCFNumberFormatterDefaultFormat;
const CFStringRef kCFNumberFormatterMultiplier;
const CFStringRef kCFNumberFormatterPositivePrefix;
const CFStringRef kCFNumberFormatterPositiveSuffix;
const CFStringRef kCFNumberFormatterNegativePrefix;
const CFStringRef kCFNumberFormatterNegativeSuffix;
const CFStringRef kCFNumberFormatterPerMillSymbol;
const CFStringRef kCFNumberFormatterInternationalCurrencySymbol;
const CFStringRef kCFNumberFormatterCurrencyGroupingSeparator;
const CFStringRef kCFNumberFormatterIsLenient;
const CFStringRef kCFNumberFormatterUseSignificantDigits;
const CFStringRef kCFNumberFormatterMinSignificantDigits;
const CFStringRef kCFNumberFormatterMaxSignificantDigits;
```

Constants

`kCFNumberFormatterCurrencyCode`

Specifies the currency code, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterDecimalSeparator`
Specifies the decimal separator, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterCurrencyDecimalSeparator`
Specifies the currency decimal separator, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterAlwaysShowDecimalSeparator`
Specifies if the result of converting a value to a string should always contain the decimal separator, even if the number is an integer.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterGroupingSeparator`
Specifies the grouping separator, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterUseGroupingSeparator`
Specifies if the grouping separator should be used, a `CFBoolean` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPercentSymbol`
Specifies the string that is used to represent the percent symbol, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterZeroSymbol`
Specifies the string that is used to represent zero, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterNaNSymbol`
Specifies the string that is used to represent NaN (“not a number”) when values are converted to strings, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterInfinitySymbol`
Specifies the string that is used to represent the symbol for infinity, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMinusSign`
Specifies the symbol for the minus sign, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPlusSign`
Specifies the symbol for the plus sign, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterCurrencySymbol`
Specifies the symbol for the currency, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterExponentSymbol`
Specifies the exponent symbol ("E" or "e") in the scientific notation of numbers (for example, as in `1.0e+56`), a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMinIntegerDigits`
Specifies the minimum number of integer digits before a decimal point, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMaxIntegerDigits`
Specifies the maximum number of integer digits before a decimal point, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMinFractionDigits`
Specifies the minimum number of digits after a decimal point, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMaxFractionDigits`
Specifies the maximum number of digits after a decimal point, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterGroupingSize`
Specifies how often the "thousands" or grouping separator appears, as in "10,000,000", a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterSecondaryGroupingSize`
Specifies how often the secondary grouping separator appears, a `CFNumber` object. See Unicode Technical Standard #35 for more information.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterRoundingMode`
Specifies how the last digit is rounded, as when `3.1415926535...` is rounded to three decimal places, as in `3.142`, a `CFNumber` object. See "Rounding Modes" (page 403) for possible values.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterRoundingIncrement`
Specifies a positive rounding increment, or `0.0` to disable rounding, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterFormatWidth`
Specifies the width of a formatted number within a string that is either left justified or right justified based on the value of `kCFNumberFormatterPaddingPosition` (page 401), a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPaddingPosition`
Specifies the position of a formatted number within a string, a `CFNumber` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPaddingCharacter`
Specifies the padding character to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterDefaultFormat`
The original format string for the formatter (given the date and time style and locale specified at creation), a `CFString` object.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMultiplier`
Specifies the multiplier to use when placing a formatted number within a string, a `CFNumber` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPositivePrefix`
Specifies the plus sign prefix symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPositiveSuffix`
Specifies the plus sign suffix symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterNegativePrefix`
Specifies the minus sign prefix symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterNegativeSuffix`
Specifies the minus sign suffix symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPerMillSymbol`
Specifies the per mill (1/1000) symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterInternationalCurrencySymbol`
Specifies the international currency symbol to use when placing a formatted number within a string, a `CFString` object.

Available in Mac OS X v10.4 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterCurrencyGroupingSeparator`
Specifies the grouping symbol to use when placing a currency value within a string, a `CFString` object.

Available in Mac OS X v10.5 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterIsLenient`
Specifies whether the formatter is lenient, a`CFBoolean` object.

Available in Mac OS X v10.5 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterUseSignificantDigits`
Specifies the whether the formatter uses significant digits, a `CFBoolean` object.

Available in Mac OS X v10.5 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMinSignificantDigits`
Specifies the minimum number of significant digits to use, a`CFNumber` object.

Available in Mac OS X v10.5 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterMaxSignificantDigits`
Specifies the maximum number of significant digits to use, a`CFNumber` object.

Available in Mac OS X v10.5 and later.

Declared in `CFNumberFormatter.h`.

Discussion
The values for these keys are all `CFType` objects. The specific types for each key are specified above.

Declared In
CFNumberFormatter.h

Number Format Options

These constants are used to specify how numbers should be parsed.

```
enum {
    kCFNumberFormatterParseIntegersOnly = 1
};
```

Constants
kCFNumberFormatterParseIntegersOnly
Specifies that only integers should be parsed.
Available in Mac OS X v10.3 and later.
Declared in CFNumberFormatter.h.

Declared In
CFNumberFormatter.h

Rounding Modes

These constants are used to specify how numbers should be rounded.

```
typedef enum {
    kCFNumberFormatterRoundCeiling = 0,
    kCFNumberFormatterRoundFloor = 1,
    kCFNumberFormatterRoundDown = 2,
    kCFNumberFormatterRoundUp = 3,
    kCFNumberFormatterRoundHalfEven = 4,
    kCFNumberFormatterRoundHalfDown = 5,
    kCFNumberFormatterRoundHalfUp = 6
} CFNumberFormatterRoundingMode;
```

Constants
kCFNumberFormatterRoundCeiling
Round up to next larger number with the proper number of fraction digits.
Available in Mac OS X v10.3 and later.
Declared in CFNumberFormatter.h.

kCFNumberFormatterRoundFloor
Round down to next larger number with the proper number of fraction digits.
Available in Mac OS X v10.3 and later.
Declared in CFNumberFormatter.h.

kCFNumberFormatterRoundDown
Round down to next larger number with the proper number of fraction digits.
Available in Mac OS X v10.3 and later.
Declared in CFNumberFormatter.h.

`kCFNumberFormatterRoundUp`
Round up to next larger number with the proper number of fraction digits.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterRoundHalfEven`
Round the last digit, when followed by a `5`, toward an even digit (`.25 -> .2, .35 -> .4`)

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterRoundHalfDown`
Round down when a `5` follows putative last digit.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterRoundHalfUp`
Round up when a 5 follows putative last digit.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

Declared In
`CFNumberFormatter.h`

Padding Positions

These constants are used to specify how numbers should be padded.

```
typedef enum {
    kCFNumberFormatterPadBeforePrefix = 0,
    kCFNumberFormatterPadAfterPrefix = 1,
    kCFNumberFormatterPadBeforeSuffix = 2,
    kCFNumberFormatterPadAfterSuffix = 3
};
```

Constants
`kCFNumberFormatterPadBeforePrefix`
Specifies the number of padding characters before the prefix.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPadAfterPrefix`
Specifies the number of padding characters after the prefix.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPadBeforeSuffix`
Specifies the number of padding characters before the suffix.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

`kCFNumberFormatterPadAfterSuffix`

Specifies the number of padding characters after the suffix.

Available in Mac OS X v10.3 and later.

Declared in `CFNumberFormatter.h`.

Declared In
`CFNumberFormatter.h`

CFPlugIn Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFPlugIn.h CFBundle.h
Companion guide	Plug-ins

Overview

`CFPlugIn` provides a standard architecture for application extensions. With `CFPlugIn`, you can design your application as a host framework that uses a set of executable code modules called plug-ins to provide certain well-defined areas of functionality. This approach allows third-party developers to add features to your application without requiring access to your source code. You can also bundle together plug-ins for multiple platforms and let `CFPlugIn` transparently load the appropriate plug-in at runtime. You can use `CFPlugIn` to add plug-in capability to, or write a plug-in for, your application.

Functions by Task

Creating Plug-ins

`CFPlugInCreate` (page 409)
Creates a CFPlugIn given its URL.

`CFPlugInInstanceCreate` (page 411)
Creates a `CFPlugIn` instance of a given type using a given factory.

Registration

`CFPlugInRegisterFactoryFunction` (page 412)
Registers a factory function and its UUID with a `CFPlugIn` object.

`CFPlugInRegisterFactoryFunctionByName` (page 413)
Registers a factory function with a `CFPlugIn` object using the function's name instead of its UUID.

`CFPlugInRegisterPlugInType` (page 413)
Registers a type and its corresponding factory function with a `CFPlugIn` object.

`CFPlugInUnregisterFactory` (page 415)
Removes the given function from a plug-in's list of registered factory functions.

`CFPlugInUnregisterPlugInType` (page 415)
Removes the given type from a plug-in's list of registered types.

CFPlugIn Miscellaneous Functions

`CFPlugInAddInstanceForFactory` (page 408)
Registers a new instance of a type with `CFPlugIn`.

`CFPlugInFindFactoriesForPlugInType` (page 409)
Searches all registered plug-ins for factory functions capable of creating an instance of the given type.

`CFPlugInFindFactoriesForPlugInTypeInPlugIn` (page 410)
Searches the given plug-in for factory functions capable of creating an instance of the given type.

`CFPlugInGetBundle` (page 410)
Returns a plug-in's bundle.

`CFPlugInGetTypeID` (page 411)
Returns the type identifier for the `CFPlugIn` opaque type.

`CFPlugInIsLoadOnDemand` (page 412)
Determines whether or not a plug-in is loaded on demand.

`CFPlugInRemoveInstanceForFactory` (page 414)
Unregisters an instance of a type with `CFPlugIn`.

`CFPlugInSetLoadOnDemand` (page 414)
Enables or disables load on demand for plug-ins that do dynamic registration (only when a client requests an instance of a supported type).

Functions

CFPlugInAddInstanceForFactory

Registers a new instance of a type with `CFPlugIn`.

```
void CFPlugInAddInstanceForFactory (
   CFUUIDRef factoryID
);
```

Parameters

factoryID
The `CFUUID` object representing the plug-in factory.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
iSpendPlugin

QuickLookSketch
Spotlight
SpotlightFortunes

Declared In
`CFPlugIn.h`

CFPlugInCreate

Creates a CFPlugIn given its URL.

```
CFPlugInRef CFPlugInCreate (
   CFAllocatorRef allocator,
   CFURLRef plugInURL
);
```

Parameters

`allocator`
The allocator to use to allocate memory for the new plug-in. Pass `NULL` or `kCFAllocatorDefault` to use the default allocator.

`plugInURL`
The location of the plug-in.

Return Value
A new plug-in. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BasicPlugIn

Declared In
`CFPlugIn.h`

CFPlugInFindFactoriesForPlugInType

Searches all registered plug-ins for factory functions capable of creating an instance of the given type.

```
CFArrayRef CFPlugInFindFactoriesForPlugInType (
   CFUUIDRef typeUUID
);
```

Parameters

`typeUUID`
A UUID type.

Return Value
An array of UUIDs for factory functions capable of creating an instance of the given type.

Availability
Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In
CFPlugIn.h

CFPlugInFindFactoriesForPlugInTypeInPlugIn

Searches the given plug-in for factory functions capable of creating an instance of the given type.

```
CFArrayRef CFPlugInFindFactoriesForPlugInTypeInPlugIn (
   CFUUIDRef typeUUID,
   CFPlugInRef plugIn
);
```

Parameters

typeUUID
A UUID type.

plugIn
The plug-in to search.

Return Value
An array of UUIDs for factory functions capable of creating an instance of the given type.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BasicPlugIn

Declared In
CFPlugIn.h

CFPlugInGetBundle

Returns a plug-in's bundle.

```
CFBundleRef CFPlugInGetBundle (
   CFPlugInRef plugIn
);
```

Parameters

plugIn
The plug-in whose bundle to obtain.

Return Value
The bundle for *plugIn*. Ownership follows the Get Rule.

Discussion
You should *always* use this function to get a plug-in's bundle. Never attempt to access the plug-in directly as a bundle.

Availability
Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInGetTypeID

Returns the type identifier for the `CFPlugIn` opaque type.

```
CFTypeID CFPlugInGetTypeID (
   void
);
```

Return Value
The type identifier for the `CFPlugIn` opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInInstanceCreate

Creates a `CFPlugIn` instance of a given type using a given factory.

```
void * CFPlugInInstanceCreate (
   CFAllocatorRef allocator,
   CFUUIDRef factoryUUID,
   CFUUIDRef typeUUID
);
```

Parameters

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

`factoryUUID`
The UUID representing the factory function to use to create a plug-in of the given type.

`typeUUID`
The UUID type.

Return Value
Returns the IUnknown interface for the new plug-in.

Discussion
The plug-in host uses this function to create an instance of the given type. Unless the plug-in is using dynamic registration, this function causes the plug-in's code to be loaded into memory.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BasicPlugIn

Declared In
`CFPlugIn.h`

CFPlugInIsLoadOnDemand

Determines whether or not a plug-in is loaded on demand.

```
Boolean CFPlugInIsLoadOnDemand (
   CFPlugInRef plugIn
);
```

Parameters

plugIn

The plug-in to query.

Return Value
`true` if the plug-in is loaded only when a client requests an instance of a supported type, otherwise `false`.

Discussion
Plug-ins that do static registration are load on demand by default. Plug-ins that do dynamic registration are not load on demand by default.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInRegisterFactoryFunction

Registers a factory function and its UUID with a `CFPlugIn` object.

```
Boolean CFPlugInRegisterFactoryFunction (
   CFUUIDRef factoryUUID,
   CFPlugInFactoryFunction func
);
```

Parameters

factoryUUID

The `CFUUID` object representing the factory function to register.

func

The factory function pointer to register.

Return Value
`true` if the factory function was successfully registered, otherwise `false`.

Discussion
This function is used by a plug-in or host when performing dynamic registration.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInRegisterFactoryFunctionByName

Registers a factory function with a `CFPlugIn` object using the function's name instead of its UUID.

```
Boolean CFPlugInRegisterFactoryFunctionByName (
   CFUUIDRef factoryUUID,
   CFPlugInRef plugIn,
   CFStringRef functionName
);
```

Parameters

`factoryUUID`
The `CFUUID` object representing the factory function to register.

`plugIn`
The plug-in containing *`functionName`*.

`functionName`
The name of the factory function to register.

Return Value
`true` if the factory function was successfully registered, otherwise `false`.

Discussion
This function is used by a plug-in or host when performing dynamic registration.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInRegisterPlugInType

Registers a type and its corresponding factory function with a `CFPlugIn` object.

```
Boolean CFPlugInRegisterPlugInType (
   CFUUIDRef factoryUUID,
   CFUUIDRef typeUUID
);
```

Parameters

`factoryUUID`
The `CFUUID` object representing the factory function that can create the type being registered.

`typeUUID`
The UUID type to register.

Return Value
`true` if the factory function was successfully registered, otherwise `false`.

Discussion
This function is used by a plug-in or host when performing dynamic registration.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInRemoveInstanceForFactory

Unregisters an instance of a type with `CFPlugIn`.

```
void CFPlugInRemoveInstanceForFactory (
   CFUUIDRef factoryID
);
```

Parameters
`factoryID`
The `CFUUID` object representing the plug-in factory.

Discussion
If the instance counts of every factory in a plug-in are zero, the plug-in can be unloaded.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
iSpendPlugin
QuickLookSketch
Spotlight
SpotlightFortunes

Declared In
`CFPlugIn.h`

CFPlugInSetLoadOnDemand

Enables or disables load on demand for plug-ins that do dynamic registration (only when a client requests an instance of a supported type).

```
void CFPlugInSetLoadOnDemand (
   CFPlugInRef plugIn,
   Boolean flag
);
```

Parameters

plugIn

The plug-in to be loaded on demand.

flag

`true` to enable load on demand, `false` otherwise.

Discussion

Plug-ins that do static registration are load on demand by default. Plug-ins that do dynamic registration are not load on demand by default.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFPlugIn.h`

CFPlugInUnregisterFactory

Removes the given function from a plug-in's list of registered factory functions.

```
Boolean CFPlugInUnregisterFactory (
   CFUUIDRef factoryUUID
);
```

Parameters

factoryUUID

The `CFUUID` object representing the factory to unregister.

Return Value

`true` if the factory function was successfully unregistered, otherwise `false`.

Discussion

Used by a plug-in or host when performing dynamic registration.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFPlugIn.h`

CFPlugInUnregisterPlugInType

Removes the given type from a plug-in's list of registered types.

```
Boolean CFPlugInUnregisterPlugInType (
   CFUUIDRef factoryUUID,
   CFUUIDRef typeUUID
);
```

Parameters

factoryUUID

The `CFUUID` object representing the factory function for the type to unregister.

typeUUID

The UUID type to unregister.

Return Value

`true` if the factory function was successfully unregistered, otherwise `false`.

Discussion

Used by a plug-in or host when performing dynamic registration.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFPlugIn.h`

Callbacks

CFPlugInDynamicRegisterFunction

A callback which provides a plug-in the opportunity to dynamically register its types with a host.

```
typedef void (*CFPlugInDynamicRegisterFunction) (
    CFPlugInRef plugIn
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFPlugInRef plugIn
);
```

Parameters

plugIn

The `CFPlugIn` object that is engaged in dynamic registration. When using in C++, this parameter functions as a `this` pointer for the plug-in.

Discussion

This callback is called as a plug-in is being loaded. This provides the plugin the means to dynamically register its types and factories with a plug-in's host. The call is triggered by the presence of `kCFPlugInDynamicRegistrationKey` (page 418) in the plug-in's information property list.

Availability

Available in Mac OS X v10.0 and later.

Declared In
CFPlugIn.h

CFPlugInFactoryFunction

Callback function that a plug-in author must implement to create a plug-in instance.

```
typedef void *(*CFPlugInFactoryFunction) (
    CFAllocatorRef allocator,
    CFUUIDRef typeUUID
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    CFAllocatorRef allocator,
    CFUUIDRef typeUUID
);
```

Parameters

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

typeUUID
The UUID type to instantiate.

Discussion

The plug-in author's implementation of this function is registered with `CFPlugIn` either statically in the plug-in's information property list, or dynamically. This function is executed as a result of a call to `CFPlugInInstanceCreate` (page 411) by the plug-in host.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFPlugIn.h

CFPlugInUnloadFunction

Callback function that is called, if present, just before a plug-in's code is unloaded.

```
typedef void (*CFPlugInUnloadFunction) (
    CFPlugInRef plugIn
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFPlugInRef plugIn
);
```

Parameters

plugIn

The `CFPlugIn` object that is about to be unloaded from memory. When writing in C++, this parameter functions as a `this` pointer for the plug-in.

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

Declared In
`CFPlugIn.h`

Data Types

CFPlugInRef

A reference to a `CFPlugin` object.

```
typedef struct __CFBundle *CFPlugInRef;
```

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CFBundle.h`

Constants

Information Property List Keys

A plug-in's information property list can contain these keys used for registering types, factories, and interfaces.

```
const CFStringRef kCFPlugInDynamicRegistrationKey;
const CFStringRef kCFPlugInDynamicRegisterFunctionKey;
const CFStringRef kCFPlugInUnloadFunctionKey;
const CFStringRef kCFPlugInFactoriesKey;
const CFStringRef kCFPlugInTypesKey;
```

Constants

`kCFPlugInDynamicRegistrationKey`

Indicates whether a plug-in requires dynamic registration.

Available in Mac OS X v10.0 and later.

Declared in `CFPlugIn.h`.

`kCFPlugInDynamicRegisterFunctionKey`

Used to specify a plug-in's registration function.

Available in Mac OS X v10.0 and later.

Declared in `CFPlugIn.h`.

`kCFPlugInUnloadFunctionKey`
Used to specify a plug-in's unload function.

Available in Mac OS X v10.0 and later.

Declared in `CFPlugIn.h`.

`kCFPlugInFactoriesKey`
Used to statically register factory functions.

Available in Mac OS X v10.0 and later.

Declared in `CFPlugIn.h`.

`kCFPlugInTypesKey`
Used to statically register the factories that can create each supported type.

Available in Mac OS X v10.0 and later.

Declared in `CFPlugIn.h`.

Availability
Mac OS X version 10.0 and later

Declared In
`CFPlugIn.h`

CFPlugInInstance Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFPlugIn.h
Companion guide	Plug-ins

Overview

CFPlugInInstance is deprecated. Use the functions defined by CFPlugIn instead.

Functions

CFPlugInInstanceCreateWithInstanceDataSize

Not recommended.

Not recommended

```
CFPlugInInstanceRef CFPlugInInstanceCreateWithInstanceDataSize (
   CFAllocatorRef allocator,
   CFIndex instanceDataSize,
   CFPlugInInstanceDeallocateInstanceDataFunction deallocateInstanceFunction,
   CFStringRef factoryName,
   CFPlugInInstanceGetInterfaceFunction getInterfaceFunction
);
```

Availability
Not recommended. Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInInstanceGetFactoryName

Not recommended.

Not recommended

```
CFStringRef CFPlugInInstanceGetFactoryName (
   CFPlugInInstanceRef instance
);
```

Availability
Not recommended. Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInInstanceGetInstanceData

Not recommended.

Not recommended

```
void * CFPlugInInstanceGetInstanceData (
   CFPlugInInstanceRef instance
);
```

Availability
Not recommended. Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInInstanceGetInterfaceFunctionTable

Not recommended.

Not recommended

```
Boolean CFPlugInInstanceGetInterfaceFunctionTable (
   CFPlugInInstanceRef instance,
   CFStringRef interfaceName,
   void **ftbl
);
```

Availability
Not recommended. Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

CFPlugInInstanceGetTypeID

Not recommended.

Not recommended

```
CFTypeID CFPlugInInstanceGetTypeID (
   void
);
```

Availability
Not recommended. Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFPlugIn.h`

Callbacks

CFPlugInInstanceDeallocateInstanceDataFunction

Not recommended.

Not recommended

```
typedef void (*CFPlugInInstanceDeallocateInstanceDataFunction) (
    void *instanceData
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *instanceData
);
```

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

Declared In
`CFPlugIn.h`

CFPlugInInstanceGetInterfaceFunction

Not recommended.

Not recommended

```
typedef Boolean (*CFPlugInInstanceGetInterfaceFunction) (
    CFPlugInInstanceRef instance,
    CFStringRef interfaceName,
    void **ftbl
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
    CFPlugInInstanceRef instance,
    CFStringRef interfaceName,
```

```
    void **ftbl
);
```

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

Declared In
`CFPlugIn.h`

Data Types

CFPlugInInstanceRef

Not recommended.

```
typedef struct __CFPlugInInstance *CFPlugInInstanceRef;
```

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

Declared In
`CFPlugIn.h`

CFPropertyList Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFPropertyList.h CFBase.h
Companion guides	Property List Programming Topics for Core Foundation XML Programming Topics for Core Foundation

Overview

CFPropertyList provides functions that convert property list objects to and from several serialized formats such as XML. The `CFPropertyListRef` (page 435) type that denotes CFPropertyList objects is an abstract type for property list objects. Depending on the contents of the XML data used to create the property list, `CFPropertyListRef` can be any of the property list objects: CFData, CFString, CFArray, CFDictionary, CFDate, CFBoolean, and CFNumber. Note that if you use a property list to generate XML, the keys of any dictionaries in the property list must be CFString objects.

It is important to understand that CFPropertyList provides an abstraction for all the property list types—you can think of CFPropertyList in object-oriented terms as being the superclass of CFString, CFNumber, CFDictionary, and so on. When a Core Foundation function returns a `CFPropertyListRef`, it means that the value may be any of the property list types. For example, `CFPreferencesCopyAppValue` (page 835) returns a `CFPropertyListRef`. This means that the value returned can be a CFString object, a CFNumber object, a CFDictionary object, and so on again. You can use `CFGetTypeID` (page 655) to determine what type of object a property list value is.

You use one of the `CFPropertyListCreate...` functions to create a property list object given an existing property list object, raw XML data (as in a file), or a stream. You can also convert a property list object to XML using the `CFPropertyListCreateXMLData` (page 431) function. You use the `CFPropertyListWriteToStream` (page 433) function to write a property list to an output stream, and validate a property list object using the `CFPropertyListIsValid` (page 432) function. CFPropertyList properly takes care of endian issues—a property list (whether represented by a stream, XML, or a CFData object) created on a PowerPC-based Macintosh is correctly interpreted on an Intel-based Macintosh, and vice versa.

For code examples illustrating how to read and write property list files, see *Property List Programming Topics for Core Foundation* and in particular Saving and Restoring Property Lists.

Functions by Task

Creating a Property List

`CFPropertyListCreateWithData` (page 430)
Creates a property list from a given CFData object.

`CFPropertyListCreateWithStream` (page 431)
Create and return a property list with a CFReadStream input.

`CFPropertyListCreateDeepCopy` (page 427)
Recursively creates a copy of a given property list.

`CFPropertyListCreateFromXMLData` (page 429)
Creates a property list using the specified XML or binary property list data.

`CFPropertyListCreateFromStream` (page 428)
Creates a property list using data from a stream.

Exporting a Property List

`CFPropertyListCreateData` (page 426)
Returns a CFData object containing a serialized representation of a given property list in a specified format.

`CFPropertyListWrite` (page 433)
Write the bytes of a serialized property list out to a stream.

`CFPropertyListCreateXMLData` (page 431)
Creates an XML representation of the specified property list.

`CFPropertyListWriteToStream` (page 433)
Writes the bytes of a property list serialization out to a stream.

Validating a Property List

`CFPropertyListIsValid` (page 432)
Determines if a property list is valid.

Functions

CFPropertyListCreateData

Returns a CFData object containing a serialized representation of a given property list in a specified format.

```
CFDataRef CFPropertyListCreateData (
   CFAllocatorRef allocator,
   CFPropertyListRef propertyList,
   CFPropertyListFormat format,
   CFOptionFlags options,
   CFErrorRef *error
);
```

Parameters

allocator

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

propertyList

The property list to write out.

format

A CFPropertyListFormat constant to specify the data format. See "Property List Formats" (page 435) for possible values.

options

This parameter is currently unused and should be set to `0`.

error

If this parameter is non-NULL, if an error occurs, on return this will contain a CFError error describing the problem. Ownership follows the Create Rule.

Return Value

A CFData object containing a serialized representation of *propertyList* in a the format specified by *format*. Ownership follows the Create Rule.

Discussion

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFPropertyList.h`

CFPropertyListCreateDeepCopy

Recursively creates a copy of a given property list.

```
CFPropertyListRef CFPropertyListCreateDeepCopy (
   CFAllocatorRef allocator,
   CFPropertyListRef propertyList,
   CFOptionFlags mutabilityOption
);
```

Parameters

allocator

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

propertyList

The property list to copy. This may be any of the standard property list objects, for example a CFArray or a CFDictionary object.

mutabilityOption
A constant that specifies the degree of mutability of the returned property list. See Property List Mutability Options (page 436) for descriptions of possible values.

Return Value
A new property list that is a copy of *propertyList*. Ownership follows the Create Rule.

Discussion
Recursively creates a copy of the given property list so nested arrays and dictionaries are copied as well as the top-most container.

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

Related Sample Code
MoreSCF

Declared In
CFPropertyList.h

CFPropertyListCreateFromStream

Creates a property list using data from a stream.

```
CFPropertyListRef CFPropertyListCreateFromStream (
   CFAllocatorRef allocator,
   CFReadStreamRef stream,
   CFIndex streamLength,
   CFOptionFlags mutabilityOption,
   CFPropertyListFormat *format,
   CFStringRef *errorString
);
```

Parameters

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

stream
The stream whose data contains the content. The stream must be opened and configured—this function simply reads bytes from the stream. The stream may contain any supported property list type (see Property List Formats (page 435)).

streamLength
The number of bytes to read. If 0, this function will read to the end of the stream.

mutabilityOption
A constant that specifies the degree of mutability for the returned property list. See Property List Mutability Options (page 436) for descriptions of possible values.

format
A constant that specifies the format of the property list. See Property List Formats (page 435) for possible values.

errorString

On return, NULL if the conversion is successful, otherwise a string that describes the nature of the error. Error messages are not localized, but may be in the future, so they are not suitable for comparison.

Pass NULL if you do not wish to receive an error string. Ownership follows the Create Rule.

Return Value
A new property list initialized with the data contained in *stream*. Ownership follows the Create Rule.

Discussion
This function simply reads bytes from *stream* starting at the current location to the end, which is expected to be the end of the property list, or up to the number of bytes specified by *streamLength* if it is not 0.

Special Considerations

Warning: This function is obsolete and will be deprecated soon. Use CFPropertyListCreateWithStream (page 431) instead.

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

Declared In
CFPropertyList.h

CFPropertyListCreateFromXMLData

Creates a property list using the specified XML or binary property list data.

```
CFPropertyListRef CFPropertyListCreateFromXMLData (
   CFAllocatorRef allocator,
   CFDataRef xmlData,
   CFOptionFlags mutabilityOption,
   CFStringRef *errorString
);
```

Parameters

allocator

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

data

The raw bytes to convert into a property list. The bytes may be the content of an XML file or of a binary property list (see Property List Formats (page 435)).

mutabilityOption

A constant that specifies the degree of mutability for the returned property list. See Property List Mutability Options (page 436) for descriptions of possible values.

errorString

On return, NULL if the conversion is successful, otherwise a string that describes the nature of the error. Error messages are not localized, but may be in the future, so they are not currently suitable for comparison.

Pass NULL if you do not wish to receive an error string. Ownership follows the Create Rule.

Return Value
A new property list if the conversion is successful, otherwise NULL. Ownership follows the Create Rule.

Special Considerations

Warning: This function is obsolete and will be deprecated soon. Use `CFPropertyListCreateWithData` (page 430) instead.

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

Related Sample Code
AudioCDSample
BetterAuthorizationSample
BSDLLCTest
HID Utilities Source
SimpleStickies

Declared In
`CFPropertyList.h`

CFPropertyListCreateWithData

Creates a property list from a given CFData object.

```
CFPropertyListRef CFPropertyListCreateWithData (
   CFAllocatorRef allocator,
   CFDataRef data,
   CFOptionFlags options,
   CFPropertyListFormat *format,
   CFErrorRef *error
);
```

Parameters

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

data
A CFData object containing a serialized representation of a property list.

options
A `CFPropertyListMutabilityOptions` (page 434) constant to specify the mutability of the returned property list—see "Property List Mutability Options" (page 436) for possible values.

format
If this parameter is non-`NULL`, on return it will be set to the format of the data. See "Property List Formats" (page 435) for possible values.

error
If this parameter is non-`NULL`, if an error occurs, on return this will contain a CFError error describing the problem. Ownership follows the Create Rule.

Return Value
A new property list created from the data in *data*. If an error occurs while parsing the data, returns `NULL`. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFPropertyList.h`

CFPropertyListCreateWithStream

Create and return a property list with a CFReadStream input.

```
CFPropertyListRef CFPropertyListCreateWithStream (
   CFAllocatorRef allocator,
   CFReadStreamRef stream,
   CFIndex streamLength,
   CFOptionFlags options,
   CFPropertyListFormat *format,
   CFErrorRef *error
);
```

Parameters

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

stream
A CFReadStream that contains a serialized representation of a property list.

streamLength
The number of bytes to read from the stream. Pass `0` to read until the end of the stream is detected.

options
A `CFPropertyListMutabilityOptions` (page 434) constant to specify the mutability of the returned property list—see "Property List Mutability Options" (page 436) for possible values.

format
If this parameter is non-`NULL`, on return it will be set to the format of the data. See "Property List Formats" (page 435) for possible values.

error
If this parameter is non-`NULL`, if an error occurs, on return this will contain a CFError error describing the problem. Ownership follows the Create Rule.

Return Value
A new property list created from the data in *stream*. If an error occurs while parsing the data, returns `NULL`. Ownership follows the Create Rule.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFPropertyList.h`

CFPropertyListCreateXMLData

Creates an XML representation of the specified property list.

```
CFDataRef CFPropertyListCreateXMLData (
   CFAllocatorRef allocator,
   CFPropertyListRef propertyList
);
```

Parameters

allocator

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

propertyList

The property list to convert. This may be any of the standard property list objects, for example a CFArray or a CFDictionary object.

Return Value

A CFData object containing the XML data. Ownership follows the Create Rule.

Special Considerations

Warning: This function is obsolete and will be deprecated soon. Use `CFPropertyListCreateData` (page 426) instead.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BetterAuthorizationSample
CFPrefTopScores
HID Calibrator
HID Config Save
SimpleStickies

Declared In

`CFPropertyList.h`

CFPropertyListIsValid

Determines if a property list is valid.

```
Boolean CFPropertyListIsValid (
   CFPropertyListRef plist,
   CFPropertyListFormat format
);
```

Parameters

plist

The property list to validate.

format

A constant that specifies the allowable format of *plist*. See Property List Formats (page 435) for possible values.

Return Value
`true` if the object graph rooted at *`plist`* is a valid property list graph—that is, the property list contains no cycles, only contains property list objects, and all dictionary keys are strings; otherwise `false`.

Discussion
The debugging library version of this function prints out some useful messages.

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

Declared In
`CFPropertyList.h`

CFPropertyListWrite

Write the bytes of a serialized property list out to a stream.

```
CFIndex CFPropertyListWrite (
   CFPropertyListRef propertyList,
   CFWriteStreamRef stream,
   CFPropertyListFormat format,
   CFOptionFlags options,
   CFErrorRef *error
);
```

Parameters

`propertyList`
The property list to write out.

`stream`
The CFWriteStream to which to write the data. The stream must be opened and configured.

`format`
A CFPropertyListFormat constant to specify the data format. See "Property List Formats" (page 435) for possible values.

`options`
This parameter is currently unused and should be set to `0`.

`error`
If this parameter is non-NULL, if an error occurs, on return this will contain a CFError error describing the problem. Ownership follows the Create Rule.

Return Value
The number of bytes written to *`stream`*. If an error occurs, returns `0`.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFPropertyList.h`

CFPropertyListWriteToStream

Writes the bytes of a property list serialization out to a stream.

```
CFIndex CFPropertyListWriteToStream (
   CFPropertyListRef propertyList,
   CFWriteStreamRef stream,
   CFPropertyListFormat format,
   CFStringRef *errorString
);
```

Parameters

propertyList

The property list to write out.

stream

The stream to write to. The stream must be opened and configured—this function simply writes bytes to the stream.

format

A constant that specifies the format used to write *propertyList*. See Property List Formats (page 435) for possible values.

errorString

On return, `NULL` if the conversion is successful, otherwise a string that describes the nature of the errors. Error messages are not localized, but may be in the future, so they are not currently suitable for comparison.

Pass `NULL` if you do not wish to receive an error string. Ownership follows the Create Rule.

Return Value

The number of bytes written, or `0` if an error occurred. If `0` is returned, *errorString* will contain an error message.

Discussion

This function leaves the stream open after reading the content. When reading a property list, this function expects the reading stream to end wherever the writing ended, so that the end of the property list data can be identified.

Special Considerations

Warning: This function is obsolete and will be deprecated soon. Use `CFPropertyListWrite` (page 433) instead.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFPropertyList.h`

Data Types

CFPropertyListMutabilityOptions

Type for flags that determine the degree of mutability of newly created property lists.

```
typedef enum CFPropertyListMutabilityOptions CFPropertyListMutabilityOptions;
```

Discussion
See "Property List Mutability Options" (page 436) for possible values.

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

Declared In
`CFPropertyList.h`

CFPropertyListRef

A reference to a CFPropertyList object.

```
typedef CFTypeRef CFPropertyListRef;
```

Discussion
This is an abstract type for property list objects. The return value of the `CFPropertyListCreateFromXMLData` function depends on the contents of the given XML data. `CFPropertyListRef` can be a reference to any of the property list objects: CFData, CFString, CFArray, CFDictionary, CFDate, CFBoolean, and CFNumber.

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

Declared In
`CFBase.h`

Constants

Property List Formats

Specifies the format of a property list.

```
enum CFPropertyListFormat {
    kCFPropertyListOpenStepFormat = 1,
    kCFPropertyListXMLFormat_v1_0 = 100,
    kCFPropertyListBinaryFormat_v1_0 = 200
};
typedef enum CFPropertyListFormat CFPropertyListFormat;
```

Constants

`kCFPropertyListOpenStepFormat`
OpenStep format (use of this format is discouraged).
Available in Mac OS X v10.2 and later.
Declared in `CFPropertyList.h`.

`kCFPropertyListXMLFormat_v1_0`
XML format version 1.0.
Available in Mac OS X v10.2 and later.
Declared in `CFPropertyList.h`.

`kCFPropertyListBinaryFormat_v1_0`
Binary format version 1.0.

Available in Mac OS X v10.2 and later.

Declared in `CFPropertyList.h`.

Property List Mutability Options

Option flags that determine the degree of mutability of newly created property lists.

```
enum CFPropertyListMutabilityOptions {
    kCFPropertyListImmutable = 0,
    kCFPropertyListMutableContainers = 1,
    kCFPropertyListMutableContainersAndLeaves = 2
};
```

Constants

`kCFPropertyListImmutable`
Specifies that the property list should be immutable.

Available in Mac OS X v10.0 and later.

Declared in `CFPropertyList.h`.

`kCFPropertyListMutableContainers`
Specifies that the property list should have mutable containers but immutable leaves.

Available in Mac OS X v10.0 and later.

Declared in `CFPropertyList.h`.

`kCFPropertyListMutableContainersAndLeaves`
Specifies that the property list should have mutable containers and mutable leaves.

Available in Mac OS X v10.0 and later.

Declared in `CFPropertyList.h`.

Reading and Writing Error Codes

Error codes for property list reading and writing functions such as `CFPropertyListCreateWithData` (page 430).

```
enum {
    kCFPropertyListReadCorruptError = 3840,
    kCFPropertyListReadUnknownVersionError = 3841,
    kCFPropertyListReadStreamError = 3842,
    kCFPropertyListWriteStreamError = 3851,
};
```

Constants

`kCFPropertyListReadCorruptError`
Signifies an error parsing a property list.

Available in Mac OS X v10.6 and later.

Declared in `CFPropertyList.h`.

`kCFPropertyListReadUnknownVersionError`
: Signifies the version number in the property list is unknown.

  Available in Mac OS X v10.6 and later.

  Declared in `CFPropertyList.h`.

`kCFPropertyListReadStreamError`
: Signifies a stream error reading a property list.

  Available in Mac OS X v10.6 and later.

  Declared in `CFPropertyList.h`.

`kCFPropertyListWriteStreamError`
: Signifies a stream error writing a property list.

  Available in Mac OS X v10.6 and later.

  Declared in `CFPropertyList.h`.

CFReadStream Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFStream.h

Overview

`CFReadStream` provides an interface for reading a byte stream either synchronously or asynchronously. You can create streams that read bytes from a block of memory, a file, or a generic socket. All streams need to be opened, using CFReadStreamOpen (page 446), before reading.

Use CFWriteStream for writing byte streams. The CFNetwork framework defines an additional type of stream for reading responses to HTTP requests.

CFReadStream is "toll-free bridged" with its Cocoa Foundation counterpart, NSInputStream. 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 `NSInputStream *` parameter, you can pass in a CFReadStreamRef, and in a function where you see a CFReadStreamRef parameter, you can pass in an `NSInputStream` instance. Note, however, that you may have either a delegate or callbacks but not both. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a Read Stream

`CFReadStreamCreateWithBytesNoCopy` (page 442)
Creates a readable stream for a block of memory.

`CFReadStreamCreateWithFile` (page 443)
Creates a readable stream for a file.

Opening and Closing a Read Stream

`CFReadStreamClose` (page 441)
Closes a readable stream.

`CFReadStreamOpen` (page 446)
Opens a stream for reading.

Reading from a Stream

`CFReadStreamRead` (page 446)
Reads data from a readable stream.

Scheduling a Read Stream

`CFReadStreamScheduleWithRunLoop` (page 447)
Schedules a stream into a run loop.

`CFReadStreamUnscheduleFromRunLoop` (page 450)
Removes a read stream from a given run loop.

Examining Stream Properties

`CFReadStreamCopyProperty` (page 442)
Returns the value of a property for a stream.

`CFReadStreamGetBuffer` (page 443)
Returns a pointer to a stream's internal buffer of unread data, if possible.

`CFReadStreamCopyError` (page 441)
Returns the error associated with a stream.

`CFReadStreamGetError` (page 444)
Returns the error status of a stream. (Deprecated. Use `CFReadStreamCopyError` (page 441) instead.)

`CFReadStreamGetStatus` (page 445)
Returns the current state of a stream.

`CFReadStreamHasBytesAvailable` (page 445)
Returns a Boolean value that indicates whether a readable stream has data that can be read without blocking.

Setting Stream Properties

`CFReadStreamSetClient` (page 448)
Assigns a client to a stream, which receives callbacks when certain events occur.

`CFReadStreamSetProperty` (page 449)
Sets the value of a property for a stream.

Getting the CFReadStream Type ID

`CFReadStreamGetTypeID` (page 445)
Returns the type identifier the `CFReadStream` opaque type.

Functions

CFReadStreamClose

Closes a readable stream.

```
void CFReadStreamClose (
   CFReadStreamRef stream
);
```

Parameters

stream

The stream to close.

Discussion

This function terminates the flow of bytes and releases any system resources required by the stream. The stream is removed from any run loops in which it was scheduled. Once closed, the stream cannot be reopened.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

CFNetworkHTTPDownload

ImageClient

Declared In

`CFStream.h`

CFReadStreamCopyError

Returns the error associated with a stream.

```
CFErrorRef CFReadStreamCopyError (
    CFReadStreamRef stream
);
```

Parameters

stream

The stream to examine.

Return Value

A CFError object that describes the current problem with stream, or `NULL` if there is no error. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStream.h`

CFReadStreamCopyProperty

Returns the value of a property for a stream.

```
CFTypeRef CFReadStreamCopyProperty (
   CFReadStreamRef stream,
   CFStringRef propertyName
);
```

Parameters

stream

The stream to examine.

propertyName

The name of the stream property to obtain. The available properties for standard Core Foundation streams are listed in *CFStream Reference*.

Return Value

The value of the property *propertyName*. Ownership follows the Create Rule.

Discussion

Each type of stream can define a set of properties that either describe or configure individual streams. A property can be any information about a stream, other than the actual data the stream handles. Examples include the headers from an HTTP transmission, the expected number of bytes, file permission information, and so on. Use `CFReadStreamSetProperty` (page 449) to modify the value of a property, although some properties are read-only.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

ImageClient

Declared In

`CFStream.h`

CFReadStreamCreateWithBytesNoCopy

Creates a readable stream for a block of memory.

```
CFReadStreamRef CFReadStreamCreateWithBytesNoCopy (
   CFAllocatorRef alloc,
   const UInt8 *bytes,
   CFIndex length,
   CFAllocatorRef bytesDeallocator
);
```

Parameters

alloc

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

bytes

The memory buffer to read. This memory must exist for the lifetime of the new stream.

length
The size of *bytes*.

bytesDeallocator
The allocator to use to deallocate *bytes* when the stream is deallocated. Pass `kCFAllocatorNull` to prevent the stream from deallocating *bytes*.

Return Value
The new read stream, or `NULL` on failure. Ownership follows the Create Rule.

Discussion
You must open the stream, using CFReadStreamOpen (page 446), before reading from it.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CFStream.h`

CFReadStreamCreateWithFile

Creates a readable stream for a file.

```
CFReadStreamRef CFReadStreamCreateWithFile (
   CFAllocatorRef alloc,
   CFURLRef fileURL
);
```

Parameters

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

fileURL
The URL of the file to read. The URL must use the file scheme.

Return Value
The new readable stream object, or `NULL` on failure. Ownership follows the Create Rule.

Discussion
You must open the stream, using CFReadStreamOpen (page 446), before reading from it.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFReadStreamGetBuffer

Returns a pointer to a stream's internal buffer of unread data, if possible.

```
const UInt8 * CFReadStreamGetBuffer (
   CFReadStreamRef stream,
   CFIndex maxBytesToRead,
   CFIndex *numBytesRead
);
```

Parameters

stream

The stream to examine.

maxBytesToRead

The maximum number of bytes to read. If greater than `0`, `maxBytesToRead` limits the number of bytes read; if `0` or less, all available bytes are read.

numBytesRead

On return, contains the length of returned buffer. If *stream* is not open or has encountered an error, *numBytesRead* is set to `-1`.

Return Value

A pointer to the internal buffer of unread data for *stream*, if possible; `NULL` otherwise. The buffer is good only until the next stream operation called on the stream. You should neither change the contents of the returned buffer nor attempt to deallocate the buffer; it is still owned by the stream. The bytes returned in the buffer are considered read from the stream.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CFStream.h`

CFReadStreamGetError

Returns the error status of a stream. (Deprecated. Use `CFReadStreamCopyError` (page 441) instead.)

```
CFStreamError CFReadStreamGetError (
   CFReadStreamRef stream
);
```

Parameters

stream

The stream to examine.

Return Value

The error status of *stream* returned in a `CFStreamError` (page 866) structure.

The error field is `0` if no error has occurred. If the error field is not `0`, the `domain` field contains a code that identifies the domain in which the value of the `error` field should be interpreted.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

Declared In

`CFStream.h`

CFReadStreamGetStatus

Returns the current state of a stream.

```
CFStreamStatus CFReadStreamGetStatus (
   CFReadStreamRef stream
);
```

Parameters

stream
The stream to examine.

Return Value

The current state of *stream*. See `CFStreamStatus` (page 867) for the list of possible states.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CFStream.h`

CFReadStreamGetTypeID

Returns the type identifier the `CFReadStream` opaque type.

```
CFTypeID CFReadStreamGetTypeID (
   void
);
```

Return Value

The type identifier for the `CFReadStream` opaque type.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

Declared In

`CFStream.h`

CFReadStreamHasBytesAvailable

Returns a Boolean value that indicates whether a readable stream has data that can be read without blocking.

```
Boolean CFReadStreamHasBytesAvailable (
   CFReadStreamRef stream
);
```

Parameters

stream
The stream to examine.

Return Value
TRUE if data can be read from *stream* without blocking, otherwise FALSE. If *stream* cannot tell if data is available without actually trying to read the data, this function returns TRUE.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CFStream.h

CFReadStreamOpen

Opens a stream for reading.

```
Boolean CFReadStreamOpen (
   CFReadStreamRef stream
);
```

Parameters
stream
The stream to open.

Return Value
TRUE if *stream* was successfully opened, FALSE otherwise. If *stream* is not in the kCFStreamStatusNotOpen (page 868) state, this function returns FALSE.

Discussion
Opening a stream causes it to reserve all the system resources it requires. If the stream can open in the background without blocking, this function always returns true. To learn when a background open operation completes, you can either schedule the stream into a run loop with CFReadStreamScheduleWithRunLoop (page 447) and wait for the stream's client (set with CFReadStreamSetClient (page 448)) to be notified or you can poll the stream using CFReadStreamGetStatus (page 445), waiting for a status of kCFStreamStatusOpen (page 868) or kCFStreamStatusError (page 868).

You do not need to wait until a stream has finished opening in the background before calling the CFReadStreamRead (page 446) function. The read operation will simply block until the open has completed.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample
CFNetworkHTTPDownload
ImageClient

Declared In
CFStream.h

CFReadStreamRead

Reads data from a readable stream.

```
CFIndex CFReadStreamRead (
   CFReadStreamRef stream,
   UInt8 *buffer,
   CFIndex bufferLength
);
```

Parameters

stream

The stream from which to read.

buffer

The buffer into which to place the data.

bufferLength

The size of *buffer* and the maximum number of bytes to read.

Return Value

The number of bytes read; `0` if the stream has reached its end; or `-1` if either the stream is not open or an error occurs.

Discussion

If *stream* is in the process of opening, this function waits until it has completed. This function blocks until at least one byte is available; it does not block until *buffer* is filled. To avoid blocking, call this function only if `CFReadStreamHasBytesAvailable` (page 445) returns `TRUE` or after the stream's client (set with `CFReadStreamSetClient` (page 448)) is notified of a `kCFStreamEventHasBytesAvailable` (page 871) event.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

CFNetworkHTTPDownload

ImageClient

Declared In

`CFStream.h`

CFReadStreamScheduleWithRunLoop

Schedules a stream into a run loop.

```
void CFReadStreamScheduleWithRunLoop (
   CFReadStreamRef stream,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

stream

The stream to schedule.

runLoop

The run loop with which to schedule *stream*.

runLoopMode
The run loop mode of *runLoop* in which to schedule *stream*.

Discussion
After scheduling *stream* with a run loop, its client (set with `CFReadStreamSetClient` (page 448)) is notified when various events happen with the stream, such as when it finishes opening, when it has bytes available, and when an error occurs. A stream can be scheduled with multiple run loops and run loop modes. Use `CFReadStreamUnscheduleFromRunLoop` (page 450) to later remove *stream* from the run loop.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample
CFNetworkHTTPDownload
ImageClient

Declared In
`CFStream.h`

CFReadStreamSetClient

Assigns a client to a stream, which receives callbacks when certain events occur.

```
Boolean CFReadStreamSetClient (
   CFReadStreamRef stream,
   CFOptionFlags streamEvents,
   CFReadStreamClientCallBack clientCB,
   CFStreamClientContext *clientContext
);
```

Parameters

stream
The stream to modify.

streamEvents
The set of events for which the client should receive callbacks. The events are listed in `CFStreamEventType` (page 871). If you pass `kCFStreamEventNone` (page 871), the current client for *stream* is removed.

clientCB
The client callback function to be called when one of the events requested in *streamEvents* occurs. If `NULL`, the current client for *stream* is removed.

clientContext
A structure holding contextual information for the stream client. The function copies the information out of the structure, so the memory pointed to by *clientContext* does not need to persist beyond the function call. If `NULL`, the current client for *stream* is removed.

Return Value
`TRUE` if the stream supports asynchronous notification, otherwise `FALSE`.

Discussion
To avoid polling and blocking, you can register a client to hear about interesting events that occur on a stream. Only one client per stream is allowed; registering a new client replaces the previous one.

Once you have set a client, you need to schedule the stream in a run loop using `CFReadStreamScheduleWithRunLoop` (page 447) so that the client can receive the asynchronous notifications. You can schedule each stream in multiple run loops (for example, if you are using a thread pool). It is the caller's responsibility to ensure that at least one of the scheduled run loops is being run, otherwise the callback cannot be called.

Although all Core Foundation streams currently support asynchronous notification, future stream types may not. If a stream does not support asynchronous notification, this function returns `false`. Typically, such streams never block for device I/O (for example, a stream reading memory) and don't benefit from asynchronous notification.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample
CFNetworkHTTPDownload
ImageClient

Declared In
`CFStream.h`

CFReadStreamSetProperty

Sets the value of a property for a stream.

```
Boolean CFReadStreamSetProperty (
   CFReadStreamRef stream,
   CFStringRef propertyName,
   CFTypeRef propertyValue
);
```

Parameters

stream
The stream to modify.

propertyName
The name of the property to set. The available properties for standard Core Foundation streams are listed in *CFStream Reference*.

propertyValue
The value to which to set the property *propertyName* for *stream*. The allowed data type of the value depends on the property being set.

Return Value
`TRUE` if *stream* recognizes and accepts the given property-value pair, otherwise `FALSE`.

Discussion
Each type of stream can define a set of properties that either describe or configure individual streams. A property can be any interesting information about a stream. Examples include the headers from an HTTP transmission, the expected number of bytes, file permission information, and so on. Properties that can be set configure the behavior of the stream and may be modifiable only at particular times, such as before the stream has been opened. (In fact, you should assume that you can set properties only before opening the stream, unless otherwise noted.) To read the value of a property use `CFReadStreamCopyProperty` (page 442), although some properties are write-only.

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

Related Sample Code
CFFTPSample
CFNetworkHTTPDownload
CocoaEcho
CocoaHTTPServer
ImageClient

Declared In
`CFStream.h`

CFReadStreamUnscheduleFromRunLoop

Removes a read stream from a given run loop.

```
void CFReadStreamUnscheduleFromRunLoop (
   CFReadStreamRef stream,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

stream
The stream to unschedule.

runLoop
The run loop from which to remove *stream*.

runLoopMode
The run loop mode of *runLoop* from which to remove *stream*.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample
CFNetworkHTTPDownload
ImageClient

Declared In
`CFStream.h`

Callbacks

CFReadStreamClientCallBack

Callback invoked when certain types of activity takes place on a readable stream.

```
typedef void (*CFReadStreamClientCallBack) (
    CFReadStreamRef stream,
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFReadStreamRef stream,
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

Parameters

stream

The stream that experienced the event *eventType*.

eventType

The event that caused the callback to be called. The possible events are listed in `CFStreamEventType` (page 871).

clientCallBackInfo

The `info` member of the `CFStreamClientContext` (page 452) structure that was used when setting the client for *stream*.

Discussion

This callback is called only for the events requested when setting the client with `CFReadStreamSetClient` (page 448).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFStream.h`

Data Types

CFReadStreamRef

A reference to a readable stream object.

```
typedef struct __CFReadStream *CFReadStreamRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFStream.h`

CFStreamClientContext

A structure that contains program-defined data and callbacks with which you can configure a stream's client behavior.

```
struct CFStreamClientContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFStreamClientContext CFStreamClientContext;
```

Fields

`version`
Version number of the structure. Must be `0`.

`info`
An arbitrary pointer to program-defined data, which can be associated with the client. This pointer is passed to the callbacks defined in the context and to the client callback function `CFReadStreamClientCallBack` (page 450).

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFStream.h`

CFRunLoop Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFRunLoop.h
Companion guide	Run Loops

Overview

A CFRunLoop object monitors sources of input to a task and dispatches control when they become ready for processing. Examples of input sources might include user input devices, network connections, periodic or time-delayed events, and asynchronous callbacks.

Three types of objects can be monitored by a run loop: sources (*CFRunLoopSource Reference*), timers (*CFRunLoopTimer Reference*), and observers (*CFRunLoopObserver Reference*). To receive callbacks when these objects need processing, you must first place these objects into a run loop with `CFRunLoopAddSource` (page 457), `CFRunLoopAddTimer` (page 458), or `CFRunLoopAddObserver` (page 456). You can later remove an object from the run loop (or invalidate it) to stop receiving its callback.

Run loops have different modes in which they can run. Each mode has its own set of objects that the run loop monitors while running in that mode. Core Foundation defines a default mode, `kCFRunLoopDefaultMode` (page 471), to hold objects that should be monitored while the application (or thread) is sitting idle. Additional modes are created automatically when an object is added to an unrecognized mode. Each run loop has its own independent set of modes.

Core Foundation also defines a special pseudo-mode `kCFRunLoopCommonModes` (page 471) to hold objects that should be shared by a set of "common" modes. A mode is added to the set of "common" modes by calling `CFRunLoopAddCommonMode` (page 456). The default mode, `kCFRunLoopDefaultMode` (page 471), is always a member of the set of common modes. The `kCFRunLoopCommonModes` (page 471) constant is never passed to `CFRunLoopRunInMode` (page 467). Each run loop has its own independent set of common modes.

There is exactly one run loop per thread. You neither create nor destroy a thread's run loop. Core Foundation automatically creates it for you as needed. You obtain the current thread's run loop with `CFRunLoopGetMain` (page 462). Call `CFRunLoopRun` (page 467) to run the current thread's run loop in the default mode until the run loop is stopped with `CFRunLoopStop` (page 468). You can also call `CFRunLoopRunInMode` (page 467) to run the current thread's run loop in a specified mode for a set period of time (or until the run loop is stopped). A run loop can only run if the requested mode has at least one source or timer to monitor.

Run loops can be run recursively. You can call `CFRunLoopRun` (page 467) or `CFRunLoopRunInMode` (page 467) from within any run loop callout and create nested run loop activations on the current thread's call stack. You are not restricted in which modes you can run from within a callout. You can create another run loop activation running in any available run loop mode, including any modes already running higher in the call stack.

Cocoa and Carbon each build upon CFRunLoop to implement their own higher-level event loop. When writing a Cocoa or Carbon application, you can add your sources, timers, and observers to their run loop objects and modes. Your objects will then get monitored as part of the regular application event loop. Use the NSRunLoop instance method `getCFRunLoop` to obtain the CFRunLoop corresponding to a Cocoa run loop. In Carbon applications, use the `GetCFRunLoopFromEventLoop` function.

Functions by Task

Getting a Run Loop

`CFRunLoopGetCurrent` (page 462)
Returns the CFRunLoop object for the current thread.

`CFRunLoopGetMain` (page 462)
Returns the main CFRunLoop object.

Starting and Stopping a Run Loop

`CFRunLoopRun` (page 467)
Runs the current thread's CFRunLoop object in its default mode indefinitely.

`CFRunLoopRunInMode` (page 467)
Runs the current thread's CFRunLoop object in a particular mode.

`CFRunLoopWakeUp` (page 469)
Wakes a waiting CFRunLoop object.

`CFRunLoopStop` (page 468)
Forces a CFRunLoop object to stop running.

`CFRunLoopIsWaiting` (page 464)
Returns a Boolean value that indicates whether the run loop is waiting for an event.

Managing Sources

`CFRunLoopAddSource` (page 457)
Adds a CFRunLoopSource object to a run loop mode.

`CFRunLoopContainsSource` (page 459)
Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopSource object.

`CFRunLoopRemoveSource` (page 465)
Removes a CFRunLoopSource object from a run loop mode.

Managing Observers

`CFRunLoopAddObserver` (page 456)
Adds a CFRunLoopObserver object to a run loop mode.

`CFRunLoopContainsObserver` (page 459)
Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopObserver object.

`CFRunLoopRemoveObserver` (page 465)
Removes a CFRunLoopObserver object from a run loop mode.

Managing Run Loop Modes

`CFRunLoopAddCommonMode` (page 456)
Adds a mode to the set of run loop common modes.

`CFRunLoopCopyAllModes` (page 461)
Returns an array that contains all the defined modes for a CFRunLoop object.

`CFRunLoopCopyCurrentMode` (page 461)
Returns the name of the mode in which a given run loop is currently running.

Managing Timers

`CFRunLoopAddTimer` (page 458)
Adds a CFRunLoopTimer object to a run loop mode.

`CFRunLoopGetNextTimerFireDate` (page 463)
Returns the time at which the next timer will fire.

`CFRunLoopRemoveTimer` (page 466)
Removes a CFRunLoopTimer object from a run loop mode.

`CFRunLoopContainsTimer` (page 460)
Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopTimer object.

Scheduling Blocks

`CFRunLoopPerformBlock` (page 464)
Enqueues a Block on a given runloop to be executed as the runloop cycles in specified modes.

Getting the CFRunLoop Type ID

`CFRunLoopGetTypeID` (page 463)
Returns the type identifier for the CFRunLoop opaque type.

Functions

CFRunLoopAddCommonMode

Adds a mode to the set of run loop common modes.

```
void CFRunLoopAddCommonMode (
   CFRunLoopRef rl,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify. Each run loop has its own independent list of modes that are in the set of common modes.

mode

The run loop mode to add to the set of common modes of *rl*.

Discussion

Sources, timers, and observers get registered to one or more run loop modes and only run when the run loop is running in one of those modes. Common modes are a set of run loop modes for which you can define a set of sources, timers, and observers that are shared by these modes. Instead of registering a source, for example, to each specific run loop mode, you can register it once to the run loop's common pseudo-mode and it will be automatically registered in each run loop mode in the common mode set. Likewise, when a mode is added to the set of common modes, any sources, timers, or observers already registered to the common pseudo-mode are added to the newly added common mode.

Once a mode is added to the set of common modes, it cannot be removed.

The Add, Contains, and Remove functions for sources, timers, and observers operate on a run loop's set of common modes when you use the constant `kCFRunLoopCommonModes` (page 471) for the run loop mode.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopCopyAllModes` (page 461)
`CFRunLoopCopyCurrentMode` (page 461)

Declared In

`CFRunLoop.h`

CFRunLoopAddObserver

Adds a CFRunLoopObserver object to a run loop mode.

```
void CFRunLoopAddObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

observer

The run loop observer to add.

mode

The run loop mode to which to add *observer*. Use the constant `kCFRunLoopCommonModes` (page 471) to add *observer* to the set of objects monitored by all the common modes.

Discussion

A run loop observer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

If *rl* already contains *observer* in *mode*, this function does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopContainsObserver` (page 459)

`CFRunLoopRemoveObserver` (page 465)

Declared In

`CFRunLoop.h`

CFRunLoopAddSource

Adds a CFRunLoopSource object to a run loop mode.

```
void CFRunLoopAddSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

source

The run loop source to add.

mode

The run loop mode to which to add *source*. Use the constant `kCFRunLoopCommonModes` (page 471) to add *source* to the set of objects monitored by all the common modes.

Discussion

If *source* is a version 0 source, this function calls the `schedule` callback function specified in the context structure for *source*. See `CFRunLoopSourceContext` for more details.

A run loop source can be registered in multiple run loops and run loop modes at the same time. When the source is signaled, whichever run loop that happens to detect the signal first will fire the source.

If *rl* already contains *source* in *mode*, this function does nothing.

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

See Also
`CFRunLoopContainsSource` (page 459)
`CFRunLoopRemoveSource` (page 465)

Related Sample Code
audioburntest
bulkerase
CFLocalServer
databurntest
ImageClient

Declared In
`CFRunLoop.h`

CFRunLoopAddTimer

Adds a CFRunLoopTimer object to a run loop mode.

```
void CFRunLoopAddTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl
The run loop to modify.

timer
The run loop timer to add.

mode
The run loop mode of *rl* to which to add *timer*. Use the constant `kCFRunLoopCommonModes` (page 471) to add *timer* to the set of objects monitored by all the common modes.

Discussion
A run loop timer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

If *rl* already contains *timer* in *mode*, this function does nothing.

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

Related Sample Code
GLUT
SampleUSBMIDIDriver

Worm

Declared In
CFRunLoop.h

CFRunLoopContainsObserver

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopObserver object.

```
Boolean CFRunLoopContainsObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

rl
The run loop to examine.

observer
The run loop observer for which to search.

mode
The run loop mode in which to search for *observer*. Use the constant kCFRunLoopCommonModes (page 471) to search for *observer* in the set of objects monitored by all the common modes.

Return Value
true if *observer* is in mode *mode* of the run loop *rl*, otherwise false.

Discussion
If *observer* was added to kCFRunLoopCommonModes (page 471), this function returns true if *mode* is either kCFRunLoopCommonModes (page 471) or any of the modes that has been added to the set of common modes.

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

See Also
CFRunLoopAddObserver (page 456)
CFRunLoopRemoveObserver (page 465)

Declared In
CFRunLoop.h

CFRunLoopContainsSource

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopSource object.

```
Boolean CFRunLoopContainsSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl

The run loop to examine.

source

The run loop source for which to search.

mode

The run loop mode of *rl* in which to search. Use the constant `kCFRunLoopCommonModes` (page 471) to search for *source* in the set of objects monitored by all the common modes.

Return Value

`true` if *source* is in mode *mode* of the run loop *rl*, otherwise `false`.

Discussion

If *source* was added to `kCFRunLoopCommonModes` (page 471), this function returns `true` if *mode* is either `kCFRunLoopCommonModes` (page 471) or any of the modes that has been added to the set of common modes.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddSource` (page 457)

`CFRunLoopRemoveSource` (page 465)

Related Sample Code

HID Manager Basics

HID Utilities Source

SampleUSBMIDIDriver

Declared In

`CFRunLoop.h`

CFRunLoopContainsTimer

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopTimer object.

```
Boolean CFRunLoopContainsTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl

The run loop to examine.

timer

The run loop timer for which to search.

mode

The run loop mode of *rl* in which to search for *timer*. Use the constant kCFRunLoopCommonModes (page 471) to search for *timer* in the set of objects monitored by all the common modes.

Return Value
`true` if *timer* is in mode *mode* of the run loop *rl*, `false` otherwise.

Discussion
If *timer* was added to kCFRunLoopCommonModes (page 471), this function returns `true` if *mode* is either kCFRunLoopCommonModes (page 471) or any of the modes that has been added to the set of common modes.

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

Declared In
`CFRunLoop.h`

CFRunLoopCopyAllModes

Returns an array that contains all the defined modes for a CFRunLoop object.

```
CFArrayRef CFRunLoopCopyAllModes (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value
An array that contains all the run loop modes defined for *rl*. Ownership follows the Create Rule.

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

See Also
CFRunLoopAddCommonMode (page 456)
CFRunLoopCopyCurrentMode (page 461)

Declared In
`CFRunLoop.h`

CFRunLoopCopyCurrentMode

Returns the name of the mode in which a given run loop is currently running.

```
CFStringRef CFRunLoopCopyCurrentMode (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value
The mode in which *rl* is currently running; `NULL` if *rl* is not running. Ownership follows the Create Rule.

Discussion
When run on the current thread's run loop, the returned value identifies the run loop mode that made the callout in which your code is currently executing.

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

See Also
`CFRunLoopAddCommonMode` (page 456)
`CFRunLoopCopyAllModes` (page 461)

Declared In
`CFRunLoop.h`

CFRunLoopGetCurrent

Returns the CFRunLoop object for the current thread.

```
CFRunLoopRef CFRunLoopGetCurrent (
    void
);
```

Return Value
Current thread's run loop. Ownership follows the Get Rule.

Discussion
Each thread has exactly one run loop associated with it.

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

See Also
`CFRunLoopGetMain` (page 462)

Related Sample Code
CFLocalServer
databurntest
DockBrowser
HID Config Save
ImageClient

Declared In
`CFRunLoop.h`

CFRunLoopGetMain

Returns the main CFRunLoop object.

```
CFRunLoopRef CFRunLoopGetMain (
    void
);
```

Return Value
The main run loop. Ownership follows the Get Rule.

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

Related Sample Code
DispatchFractal

Declared In
`CFRunLoop.h`

CFRunLoopGetNextTimerFireDate

Returns the time at which the next timer will fire.

```
CFAbsoluteTime CFRunLoopGetNextTimerFireDate (
   CFRunLoopRef rl,
   CFStringRef mode
);
```

Parameters

rl
The run loop to examine.

mode
The run loop mode within *rl* to test.

Return Value
The earliest firing time of the run loop timers registered in *mode* for the run loop *rl*.

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

Declared In
`CFRunLoop.h`

CFRunLoopGetTypeID

Returns the type identifier for the CFRunLoop opaque type.

```
CFTypeID CFRunLoopGetTypeID (
   void
);
```

Return Value
The type identifier for the CFRunLoop opaque type.

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

Related Sample Code
CFLocalServer

Declared In
`CFRunLoop.h`

CFRunLoopIsWaiting

Returns a Boolean value that indicates whether the run loop is waiting for an event.

```
Boolean CFRunLoopIsWaiting (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value
`true` if *rl* has no events to process and is blocking, waiting for a source or timer to become ready to fire; `false` if *rl* either is not running or is currently processing a source, timer, or observer.

Discussion
This function is useful only to test the state of another thread's run loop. When called with the current thread's run loop, this function always returns `false`.

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

Declared In
`CFRunLoop.h`

CFRunLoopPerformBlock

Enqueues a Block on a given runloop to be executed as the runloop cycles in specified modes.

```
void CFRunLoopPerformBlock (
    CFRunLoopRef rl,
    CFTypeRef mode,
    void (^block)(void)
);
```

Parameters

rl

A run loop.

mode

A CFString that identifies a runloop mode, or a CFArray of CFStrings that each identify a runloop mode.

block

The Block to execute.

The Block is copied by the function before the function returns.

Discussion
When the runloop is run in the mode or one of the modes specified by *mode*, the Block is executed. You can use this function as a means to get work "onto" another thread (similar to Cocoa's `performSelector:onThread:withObject:waitUntilDone:` and related methods) as an alternative to mechanisms such as putting a CFRunLoopTimer in the other thread's run loop, or using CFMessagePort to pass information between threads.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFRunLoop.h`

CFRunLoopRemoveObserver

Removes a CFRunLoopObserver object from a run loop mode.

```
void CFRunLoopRemoveObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

rl
The run loop to modify.

observer
The run loop observer to remove.

mode
The run loop mode of *rl* from which to remove *observer*. Use the constant `kCFRunLoopCommonModes` (page 471) to remove *observer* from the set of objects monitored by all the common modes.

Discussion
If *rl* does not contain *observer* in *mode*, this function does nothing.

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

See Also
`CFRunLoopAddObserver` (page 456)
`CFRunLoopContainsObserver` (page 459)

Declared In
`CFRunLoop.h`

CFRunLoopRemoveSource

Removes a CFRunLoopSource object from a run loop mode.

```
void CFRunLoopRemoveSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

source

The run loop source to remove.

mode

The run loop mode of *rl* from which to remove *source*. Use the constant `kCFRunLoopCommonModes` (page 471) to remove *source* from the set of objects monitored by all the common modes.

Discussion

If *source* is a version 0 source, this function calls the `cancel` callback function specified in the context structure for *source*. See `CFRunLoopSourceContext` and `CFRunLoopSourceContext1`for more details.

If *rl* does not contain *source* in *mode*, this function does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddSource` (page 457)

`CFRunLoopContainsSource` (page 459)

Related Sample Code

BackgroundExporter

DNSServiceMetaQuery

HID Manager Basics

iChatStatusFromApplication

SampleUSBMIDIDriver

Declared In

`CFRunLoop.h`

CFRunLoopRemoveTimer

Removes a CFRunLoopTimer object from a run loop mode.

```
void CFRunLoopRemoveTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

timer
The run loop timer to remove.

mode
The run loop mode of *rl* from which to remove *timer*. Use the constant `kCFRunLoopCommonModes` (page 471) to remove *timer* from the set of objects monitored by all the common modes.

Discussion
If *rl* does not contain *timer* in *mode*, this function does nothing.

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

Related Sample Code
GLUT
SampleUSBMIDIDriver

Declared In
`CFRunLoop.h`

CFRunLoopRun

Runs the current thread's CFRunLoop object in its default mode indefinitely.

```
void CFRunLoopRun (
   void
);
```

Discussion
The current thread's run loop runs in the default mode (see "Default Run Loop Mode" (page 471)) until the run loop is stopped with `CFRunLoopStop` (page 468) or all the sources and timers are removed from the default run loop mode.

Run loops can be run recursively. You can call `CFRunLoopRun` from within any run loop callout and create nested run loop activations on the current thread's call stack.

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

Related Sample Code
audioburntest
bulkerase
CFLocalServer
databurntest
STUCAuthoringDeviceTool

Declared In
`CFRunLoop.h`

CFRunLoopRunInMode

Runs the current thread's CFRunLoop object in a particular mode.

```
SInt32 CFRunLoopRunInMode (
   CFStringRef mode,
   CFTimeInterval seconds,
   Boolean returnAfterSourceHandled
);
```

Parameters

mode

The run loop mode to run. *mode* can be any arbitrary CFString. You do not need to explicitly create a run loop mode, although a run loop mode needs to contain at least one source or timer to run.

seconds

The length of time to run the run loop. If `0`, only one pass is made through the run loop before returning; if multiple sources or timers are ready to fire immediately, only one (possibly two if one is a version 0 source) will be fired, regardless of the value of *returnAfterSourceHandled*.

returnAfterSourceHandled

A flag indicating whether the run loop should exit after processing one source. If `false`, the run loop continues processing events until *seconds* has passed.

Return Value

A value indicating the reason the run loop exited. Possible values are described below.

Discussion

Run loops can be run recursively. You can call `CFRunLoopRunInMode` from within any run loop callout and create nested run loop activations on the current thread's call stack. You are not restricted in which modes you can run from within a callout. You can create another run loop activation running in any available run loop mode, including any modes already running higher in the call stack.

The run loop exits with the following return values under the indicated conditions:

- `kCFRunLoopRunFinished`. The run loop mode *mode* has no sources or timers.
- `kCFRunLoopRunStopped`. The run loop was stopped with `CFRunLoopStop` (page 468).
- `kCFRunLoopRunTimedOut`. The time interval *seconds* passed.
- `kCFRunLoopRunHandledSource`. A source was processed. This exit condition only applies when *returnAfterSourceHandled* is `true`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioQueueTools
DefaultOutputUnit
FSFileOperation
HID Manager Basics
STUCAuthoringDeviceCocoaSample

Declared In

`CFRunLoop.h`

CFRunLoopStop

Forces a CFRunLoop object to stop running.

```
void CFRunLoopStop (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to stop.

Discussion

This function forces *rl* to stop running and return control to the function that called `CFRunLoopRun` (page 467) or `CFRunLoopRunInMode` (page 467) for the current run loop activation. If the run loop is nested with a callout from one activation starting another activation running, only the innermost activation is exited.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

audioburntest

bulkerase

CFFTPSample

CFLocalServer

databurntest

Declared In

`CFRunLoop.h`

CFRunLoopWakeUp

Wakes a waiting CFRunLoop object.

```
void CFRunLoopWakeUp (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to wake up.

Discussion

A run loop goes to sleep when it is waiting for a source or timer to become ready to fire. If no source or timer fires, the run loop stays there until it times out or is explicitly woken up. If a run loop is modified, such as a new source added, you need to wake up the run loop to allow it to process the change. Version 0 sources use `CFRunLoopWakeUp` to cause the run loop to wake up after setting a source to be signaled, if they want the source handled immediately.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

Data Types

CFRunLoopRef

A reference to a run loop object.

```
typedef struct __CFRunLoop *CFRunLoopRef;
```

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

Declared In
`CFRunLoop.h`

Constants

CFRunLoopRunInMode Exit Codes

Return codes for `CFRunLoopRunInMode`, identifying the reason the run loop exited.

```
enum {
    kCFRunLoopRunFinished = 1,
    kCFRunLoopRunStopped = 2,
    kCFRunLoopRunTimedOut = 3,
    kCFRunLoopRunHandledSource = 4
};
```

Constants

`kCFRunLoopRunFinished`
The running run loop mode has no sources or timers to process.
Available in Mac OS X v10.0 and later.
Declared in `CFRunLoop.h`.

`kCFRunLoopRunStopped`
`CFRunLoopStop` (page 468) was called on the run loop.
Available in Mac OS X v10.0 and later.
Declared in `CFRunLoop.h`.

`kCFRunLoopRunTimedOut`
The specified time interval for running the run loop has passed.
Available in Mac OS X v10.0 and later.
Declared in `CFRunLoop.h`.

`kCFRunLoopRunHandledSource`
A source has been processed. This value is returned only if the run loop was told to run only until a source was processed.
Available in Mac OS X v10.0 and later.
Declared in `CFRunLoop.h`.

Common Mode Flag

A run loop pseudo-mode that manages objects monitored in the “common” modes.

```
const CFStringRef kCFRunLoopCommonModes;
```

Constants

`kCFRunLoopCommonModes`

Objects added to a run loop using this value as the mode are monitored by all run loop modes that have been declared as a member of the set of “common” modes with `CFRunLoopAddCommonMode` (page 456).

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

Discussion

Run loops never run in this mode. This pseudo-mode is used only as a special set of sources, timers, and observers that is shared by other modes. See “Managing Observers” (page 455) for more details.

Default Run Loop Mode

Default run loop mode.

```
const CFStringRef kCFRunLoopDefaultMode;
```

Constants

`kCFRunLoopDefaultMode`

Run loop mode that should be used when a thread is in its default, or idle, state, waiting for an event. This mode is used when the run loop is started with `CFRunLoopRun` (page 467).

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

CFRunLoopObserver Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFRunLoop.h
Companion guide	Run Loops

Overview

A CFRunLoopObserver provides a general means to receive callbacks at different points within a running run loop. In contrast to sources, which fire when an asynchronous event occurs, and timers, which fire when a particular time passes, observers fire at special locations within the execution of the run loop, such as before sources are processed or before the run loop goes to sleep, waiting for an event to occur. Observers can be either one-time events or repeated every time through the run loop's loop.

Each run loop observer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

Functions

CFRunLoopObserverCreate

Creates a CFRunLoopObserver object.

```
CFRunLoopObserverRef CFRunLoopObserverCreate (
   CFAllocatorRef allocator,
   CFOptionFlags activities,
   Boolean repeats,
   CFIndex order,
   CFRunLoopObserverCallBack callout,
   CFRunLoopObserverContext *context
);
```

Parameters

allocator

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

activities
: Set of flags identifying the activity stages of the run loop during which the observer should be called. See Run Loop Activities (page 479)for the list of stages. To have the observer called at multiple stages in the run loop, combine the Run Loop Activities (page 479) values using the bitwise-OR operator.

repeats
: A flag identifying whether the observer should be called only once or every time through the run loop. If *repeats* is `false`, the observer is invalidated after it is called once, even if the observer was scheduled to be called at multiple stages within the run loop.

order
: A priority index indicating the order in which run loop observers are processed. When multiple run loop observers are scheduled in the same activity stage in a given run loop mode, the observers are processed in increasing order of this parameter. Pass 0 unless there is a reason to do otherwise.

callout
: The callback function invoked when the observer runs.

context
: A structure holding contextual information for the run loop observer. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be `NULL` if the observer does not need the context's `info` pointer to keep track of state.

Return Value
The new CFRunLoopObserver object. Ownership follows the Create Rule.

Discussion
The run loop observer is not automatically added to a run loop. To add the observer to a run loop, use `CFRunLoopAddObserver` (page 456). An observer can be registered to only one run loop, although it can be added to multiple run loop modes within that run loop.

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

Declared In
`CFRunLoop.h`

CFRunLoopObserverDoesRepeat

Returns a Boolean value that indicates whether a CFRunLoopObserver repeats.

```
Boolean CFRunLoopObserverDoesRepeat (
   CFRunLoopObserverRef observer
);
```

Parameters

observer
: The run loop observer to examine.

Return Value
`true` if *observer* is processed during every pass through the run loop; `false` if *observer* is processed once and then is invalidated.

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

Declared In
CFRunLoop.h

CFRunLoopObserverGetActivities

Returns the run loop stages during which an observer runs.

```
CFOptionFlags CFRunLoopObserverGetActivities (
   CFRunLoopObserverRef observer
);
```

Parameters

observer
The run loop observer to examine.

Return Value
A bitwise-OR combination of all the run loop stages in which *observer* is called. See Run Loop Activities (page 479) for the list of stages.

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

Declared In
CFRunLoop.h

CFRunLoopObserverGetContext

Returns the context information for a CFRunLoopObserver object.

```
void CFRunLoopObserverGetContext (
   CFRunLoopObserverRef observer,
   CFRunLoopObserverContext *context
);
```

Parameters

observer
The run loop observer to examine.

context
Upon return, contains the context information for *observer*. This is the same information passed to `CFRunLoopObserverCreate` (page 473) when creating *observer*.

Discussion
The context version number for run loop observers is currently `0`. Before calling this function, you need to initialize the `version` member of *context* to `0`.

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

Declared In
CFRunLoop.h

CFRunLoopObserverGetOrder

Returns the ordering parameter for a CFRunLoopObserver object.

```
CFIndex CFRunLoopObserverGetOrder (
   CFRunLoopObserverRef observer
);
```

Parameters

observer

The run loop observer to examine.

Return Value

The ordering parameter for *observer*. When multiple observers are scheduled in the same run loop mode and stage, this value determines the order (from small to large) in which the observers are called.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFRunLoop.h

CFRunLoopObserverGetTypeID

Returns the type identifier for the CFRunLoopObserver opaque type.

```
CFTypeID CFRunLoopObserverGetTypeID (
   void
);
```

Return Value

The type identifier for the CFRunLoopObserver opaque type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFRunLoop.h

CFRunLoopObserverInvalidate

Invalidates a CFRunLoopObserver object, stopping it from ever firing again.

```
void CFRunLoopObserverInvalidate (
   CFRunLoopObserverRef observer
);
```

Parameters

observer

The run loop observer to invalidate.

Discussion

Once invalidated, *observer* will never fire and call its callback function again. This function automatically removes *observer* from all run loop modes in which it had been added. The memory is not deallocated unless the run loop held the only reference to *observer*.

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

Declared In
`CFRunLoop.h`

CFRunLoopObserverIsValid

Returns a Boolean value that indicates whether a CFRunLoopObserver object is valid and able to fire.

```
Boolean CFRunLoopObserverIsValid (
   CFRunLoopObserverRef observer
);
```

Parameters

observer

The run loop observer to examine.

Return Value
`true` if *observer* is valid, otherwise `false`.

Discussion
A nonrepeating observer is automatically invalidated after it is called once.

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

Declared In
`CFRunLoop.h`

Callbacks

CFRunLoopObserverCallBack

Callback invoked when a CFRunLoopObserver object is fired.

```
typedef void (*CFRunLoopObserverCallBack) (
    CFRunLoopObserverRef observer,
    CFRunLoopActivity activity,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFRunLoopObserverRef observer,
    CFRunLoopActivity activity,
    void *info
);
```

Parameters

observer
The run loop observer that is firing.

activity
The current activity stage of the run loop.

info
The `info` member of the `CFRunLoopObserverContext` (page 478) structure that was used when creating the run loop observer.

Discussion
You specify this callback when you create the run loop observer with `CFRunLoopObserverCreate` (page 473).

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

Declared In
`CFRunLoop.h`

Data Types

CFRunLoopObserverContext

A structure that contains program-defined data and callbacks with which you can configure a CFRunLoopObserver object's behavior.

```
struct CFRunLoopObserverContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFRunLoopObserverContext CFRunLoopObserverContext;
```

Fields

`version`
Version number of the structure. Must be `0`.

`info`
An arbitrary pointer to program-defined data, which can be associated with the run loop observer at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

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

Declared In
CFRunLoop.h

CFRunLoopObserverRef

A reference to a run loop observer object.

```
typedef struct __CFRunLoopObserver *CFRunLoopObserverRef;
```

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

Declared In
CFRunLoop.h

Constants

Run Loop Activities

Run loop activity stages in which run loop observers can be scheduled.

```
enum CFRunLoopActivity {
    kCFRunLoopEntry = (1 << 0),
    kCFRunLoopBeforeTimers = (1 << 1),
    kCFRunLoopBeforeSources = (1 << 2),
    kCFRunLoopBeforeWaiting = (1 << 5),
    kCFRunLoopAfterWaiting = (1 << 6),
    kCFRunLoopExit = (1 << 7),
    kCFRunLoopAllActivities = 0x0FFFFFFFU
};
typedef enum CFRunLoopActivity CFRunLoopActivity;
```

Constants

`kCFRunLoopEntry`
The entrance of the run loop, before entering the event processing loop. This activity occurs once for each call to `CFRunLoopRun` (page 467) and `CFRunLoopRunInMode` (page 467).

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopBeforeTimers`
Inside the event processing loop before any timers are processed.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopBeforeSources`
Inside the event processing loop before any sources are processed.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopBeforeWaiting`
: Inside the event processing loop before the run loop sleeps, waiting for a source or timer to fire. This activity does not occur if `CFRunLoopRunInMode` (page 467) is called with a timeout of 0 seconds. It also does not occur in a particular iteration of the event processing loop if a version 0 source fires.

  Available in Mac OS X v10.0 and later.

  Declared in `CFRunLoop.h`.

`kCFRunLoopAfterWaiting`
: Inside the event processing loop after the run loop wakes up, but before processing the event that woke it up. This activity occurs only if the run loop did in fact go to sleep during the current loop.

  Available in Mac OS X v10.0 and later.

  Declared in `CFRunLoop.h`.

`kCFRunLoopExit`
: The exit of the run loop, after exiting the event processing loop. This activity occurs once for each call to `CFRunLoopRun` (page 467) and `CFRunLoopRunInMode` (page 467).

  Available in Mac OS X v10.0 and later.

  Declared in `CFRunLoop.h`.

`kCFRunLoopAllActivities`
: A combination of all the preceding stages.

  Available in Mac OS X v10.0 and later.

  Declared in `CFRunLoop.h`.

Discussion
The run loop stages in which an observer is scheduled are selected when the observer is created with `CFRunLoopObserverCreate` (page 473).

CFRunLoopSource Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFRunLoop.h
Companion guide	Run Loops

Overview

A CFRunLoopSource object is an abstraction of an input source that can be put into a run loop. Input sources typically generate asynchronous events, such as messages arriving on a network port or actions performed by the user.

An input source type normally defines an API for creating and operating on objects of the type, as if it were a separate entity from the run loop, then provides a function to create a CFRunLoopSource for an object. The run loop source can then be registered with the run loop and act as an intermediary between the run loop and the actual input source type object. Examples of input sources include CFMachPort, CFMessagePort, and CFSocket.

There are two categories of sources. Version 0 sources, so named because the `version` field of their context structure is 0, are managed manually by the application. When a source is ready to fire, some part of the application, perhaps code on a separate thread waiting for an event, must call `CFRunLoopSourceSignal` (page 485) to tell the run loop that the source is ready to fire. The run loop source for CFSocket is currently implemented as a version 0 source.

Version 1 sources are managed by the run loop and kernel. These sources use Mach ports to signal when the sources are ready to fire. A source is automatically signaled by the kernel when a message arrives on the source's Mach port. The contents of the message are given to the source to process when the source is fired. The run loop sources for CFMachPort and CFMessagePort are currently implemented as version 1 sources.

When creating your own custom run loop source, you can choose which version works best for you.

A run loop source can be registered in multiple run loops and run loop modes at the same time. When the source is signaled, whichever run loop that happens to detect the signal first will fire the source. Adding a source to multiple threads' run loops can be used to manage a pool of "worker" threads that is processing discrete sets of data, such as client-server messages over a network or entries in a job queue filled by a "manager" thread. As messages arrive or jobs get added to the queue, the source gets signaled and a random thread receives and processes the request.

Functions

CFRunLoopSourceCreate

Creates a CFRunLoopSource object.

```
CFRunLoopSourceRef CFRunLoopSourceCreate (
   CFAllocatorRef allocator,
   CFIndex order,
   CFRunLoopSourceContext *context
);
```

Parameters

allocator

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

order

A priority index indicating the order in which run loop sources are processed. When multiple run loop sources are firing in a single pass through the run loop, the sources are processed in increasing order of this parameter. If the run loop is set to process only one source per loop, only the highest priority source, the one with the lowest *order* value, is processed. This value is ignored for version 1 sources. Pass 0 unless there is a reason to do otherwise.

context

A structure holding contextual information for the run loop source. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call.

Return Value

The new CFRunLoopSource object. You are responsible for releasing this object.

Discussion

The run loop source is not automatically added to a run loop. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SimpleAudioExtraction

simpleJavaLauncher

Declared In

`CFRunLoop.h`

CFRunLoopSourceGetContext

Returns the context information for a CFRunLoopSource object.

```
void CFRunLoopSourceGetContext (
   CFRunLoopSourceRef source,
   CFRunLoopSourceContext *context
);
```

Parameters

source

The run loop source to examine.

context

A pointer to the structure into which the context information for *source* is to be copied. The information being returned is the same information passed to `CFRunLoopSourceCreate` (page 482) when creating *source*.

Discussion

Run loop sources come in two versions with different-sized context structures. *context* must point to the correct version of the structure for *source*. Before calling this function, you need to initialize the `version` member of *context* with the version number (either 0 or 1) of *source*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopSourceGetOrder

Returns the ordering parameter for a CFRunLoopSource object.

```
CFIndex CFRunLoopSourceGetOrder (
   CFRunLoopSourceRef source
);
```

Parameters

source

The run loop source to examine.

Return Value

The ordering parameter for *source*, which the run loop uses (for version 0 sources only) to determine the order in which sources are processed when multiple sources are firing.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopSourceGetTypeID

Returns the type identifier of the CFRunLoopSource opaque type.

```
CFTypeID CFRunLoopSourceGetTypeID (
   void
);
```

Return Value
The type identifier for the CFRunLoopSource opaque type.

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

Declared In
`CFRunLoop.h`

CFRunLoopSourceInvalidate

Invalidates a CFRunLoopSource object, stopping it from ever firing again.

```
void CFRunLoopSourceInvalidate (
   CFRunLoopSourceRef source
);
```

Parameters

source
The run loop source to invalidate.

Discussion
Once invalidated, *source* will never fire and call its perform callback function again. This function automatically removes *source* from all the run loop modes in which it was registered. If *source* is a version 0 source, this function calls its `cancel` callback function as it is removed from each run loop mode. The memory for *source* is not deallocated unless the run loop held the only reference to *source*.

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

Related Sample Code
audioburntest
bulkerase
CFLocalServer
databurntest
ImageClient

Declared In
`CFRunLoop.h`

CFRunLoopSourceIsValid

Returns a Boolean value that indicates whether a CFRunLoopSource object is valid and able to fire.

```
Boolean CFRunLoopSourceIsValid (
   CFRunLoopSourceRef source
);
```

Parameters

source

The run loop source to examine.

Return Value

`true` if *source* is valid, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopSourceSignal

Signals a CFRunLoopSource object, marking it as ready to fire.

```
void CFRunLoopSourceSignal (
   CFRunLoopSourceRef source
);
```

Parameters

source

The run loop source to signal.

Discussion

This function has no effect on version 1 sources, which are automatically handled when Mach messages arrive for them. After signaling a version 0 source, you need to call `CFRunLoopWakeUp` (page 469) on one of the run loops in which the source is registered to get the source handled immediately.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SimpleAudioExtraction

Declared In

`CFRunLoop.h`

Callbacks

CFRunLoopCancelCallBack

Callback invoked when a version 0 CFRunLoopSource object is removed from a run loop mode.

```
typedef void (*CFRunLoopCancelCallBack) (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

Parameters

info

The `info` member of the `CFRunLoopSourceContext` (page 490) structure that was used when creating the run loop source.

rl

The run loop from which the run loop source is being removed.

mode

The run loop mode from which the run loop source is being removed.

Discussion

You specify this callback in the `CFRunLoopSourceContext` (page 490) structure when creating the run loop source.

CFRunLoopEqualCallBack

Callback invoked to test two CFRunLoopSource objects for equality.

```
typedef Boolean (*CFRunLoopEqualCallBack) (
    const void *info1,
    const void *info2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
    const void *info1,
    const void *info2
);
```

Parameters

info1

The `info` member of the `CFRunLoopSourceContext` (page 490) or `CFRunLoopSourceContext1` (page 491) structure that was used when creating the first run loop source to test.

info2

The `info` member of the `CFRunLoopSourceContext` (page 490) or `CFRunLoopSourceContext1` (page 491) structure that was used when creating the second run loop source to test.

Return Value
`true` if *info1* and *info2* should be considered equal; otherwise `false`.

Discussion
You specify this callback in the `CFRunLoopSourceContext` (page 490) or `CFRunLoopSourceContext1` (page 491) structure when creating the run loop source.

CFRunLoopGetPortCallBack

Callback invoked to obtain the native Mach port represented by a version 1 CFRunLoopSource object.

```
typedef mach_port_t (*CFRunLoopGetPortCallBack) (
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
mach_port_t MyCallBack (
    void *info
);
```

Parameters
info
The `info` member of the `CFRunLoopSourceContext1` (page 491) structure that was used when creating the run loop source.

Return Value
The native Mach port for the run loop source.

Discussion
This callback is called whenever the run loop needs a source's Mach port, which can happen in each iteration of the run loop's loop. Because of the frequency with which the run loop may call this callback, make the function as efficient as possible.

A version 1 run loop source must have a one-to-one relationship between itself and its Mach port. Each source must have only one Mach port associated with it and each Mach port must represent only one source.

You specify this callback in the `CFRunLoopSourceContext1` (page 491) structure when creating the run loop source.

CFRunLoopHashCallBack

Callback invoked to compute a hash code for the `info` pointer of a CFRunLoopSource object.

```
typedef CFHashCode (*CFRunLoopHashCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFHashCode MyCallBack (
    const void *info
);
```

Parameters

info

The `info` member of the `CFRunLoopSourceContext` (page 490) or `CFRunLoopSourceContext1` (page 491) structure that was used when creating the run loop source.

Return Value

A hash code value for *info*.

Discussion

If a hash callback is not provided for a source, the `info` pointer is used.

You specify this callback in the `CFRunLoopSourceContext` (page 490) or `CFRunLoopSourceContext1` (page 491) structure when creating the run loop source.

CFRunLoopMachPerformCallBack

Callback invoked to process and optionally reply to a message received on a version 1 CFRunLoopSource object (Mach port-based sources).

```
typedef void *(*CFRunLoopMachPerformCallBack) (
    void *msg,
    CFIndex size,
    CFAllocatorRef allocator,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    void *msg,
    CFIndex size,
    CFAllocatorRef allocator,
    void *info
);
```

Parameters

msg

The Mach message received on the Mach port. The pointer is to a `mach_msg_header_t` structure. A version 0 format trailer (`mach_msg_format_0_trailer_t`) is at the end of the Mach message.

size

Size of the Mach message in *msg*, excluding the message trailer.

allocator

The allocator object that should be used to allocate a reply message.

info

The `info` member of the `CFRunLoopSourceContext1` (page 491) structure that was used when creating the run loop source.

Return Value

An optional Mach message to be sent in response to the received message. The message must be allocated using *allocator*. Return `NULL` if you want an empty reply returned to the sender.

Discussion
You only need to provide this callback if you create your own version 1 run loop source. CFMachPort and CFMessagePort run loop sources already implement this callback to forward the received message to the CFMachPort's or CFMessagePort's own callback function, which you do need to implement.

You specify this callback in the `CFRunLoopSourceContext1` (page 491) structure when creating the run loop source.

CFRunLoopPerformCallBack

Callback invoked when a message is received on a version 0 CFRunLoopSource object.

```
typedef void (*CFRunLoopPerformCallBack) (
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info
);
```

Parameters
info
The `info` member of the `CFRunLoopSourceContext` (page 490) structure that was used when creating the run loop source.

Discussion
You only need to provide this callback if you create your own version 0 run loop source. CFSocket run loop sources already implement this callback to forward the received message to the CFSocket's own callback function, which you do need to implement.

You specify this callback in the `CFRunLoopSourceContext` (page 490) structure when creating the run loop source.

CFRunLoopScheduleCallBack

Callback invoked when a version 0 CFRunLoopSource object is added to a run loop mode.

```
typedef void (*CFRunLoopScheduleCallBack) (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

Parameters

info

The `info` member of the `CFRunLoopSourceContext` (page 490) structure that was used when creating the run loop source.

rl

The run loop in which the source is being scheduled.

mode

The run loop mode in which the source is being scheduled.

Discussion

You specify this callback in the `CFRunLoopSourceContext` (page 490) structure when creating the run loop source.

Data Types

CFRunLoopSourceContext

A structure that contains program-defined data and callbacks with which you can configure a version 0 CFRunLoopSource's behavior.

```
struct CFRunLoopSourceContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFRunLoopEqualCallBack equal;
    CFRunLoopHashCallBack hash;
    CFRunLoopScheduleCallBack schedule;
    CFRunLoopCancelCallBack cancel;
    CFRunLoopPerformCallBack perform;
};
typedef struct CFRunLoopSourceContext CFRunLoopSourceContext;
```

Fields

`version`

Version number of the structure. Must be 0.

`info`

An arbitrary pointer to program-defined data, which can be associated with the CFRunLoopSource at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`

A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`

A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`

A copy description callback for your program-defined `info` pointer. Can be `NULL`.

`equal`

An equality test callback for your program-defined `info` pointer. Can be `NULL`.

`hash`
A hash calculation callback for your program-defined `info` pointer. Can be `NULL`.

`schedule`
A scheduling callback for the run loop source. This callback is called when the source is added to a run loop mode. Can be `NULL`.

`cancel`
A cancel callback for the run loop source. This callback is called when the source is removed from a run loop mode. Can be `NULL`.

`perform`
A perform callback for the run loop source. This callback is called when the source has fired.

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

Declared In
`CFRunLoop.h`

CFRunLoopSourceContext1

A structure that contains program-defined data and callbacks with which you can configure a version 1 CFRunLoopSource's behavior.

```
struct CFRunLoopSourceContext1 {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFRunLoopEqualCallBack equal;
    CFRunLoopHashCallBack hash;
    CFRunLoopGetPortCallBack getPort;
    CFRunLoopMachPerformCallBack perform;
};
typedef struct CFRunLoopSourceContext1 CFRunLoopSourceContext1;
```

Fields
`version`
Version number of the structure. Must be 1.

`info`
An arbitrary pointer to program-defined data, which can be associated with the run loop source at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

`equal`
An equality test callback for your program-defined `info` pointer. Can be `NULL`.

`hash`
: A hash calculation callback for your program-defined `info` pointer. Can be `NULL`.

`getPort`
: A callback to retrieve the native Mach port represented by the source. This callback is called when the source is either added to or removed from a run loop mode.

`perform`
: A perform callback for the run loop source. This callback is called when the source has fired.

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

Declared In
`CFRunLoop.h`

CFRunLoopSourceRef

A reference to a run loop source object.

```
typedef struct __CFRunLoopSource *CFRunLoopSourceRef;
```

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

Declared In
`CFRunLoop.h`

CFRunLoopTimer Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFRunLoop.h
Companion guide	Run Loops

Overview

A CFRunLoopTimer object represents a specialized run loop source that fires at a preset time in the future. Timers can fire either only once or repeatedly at fixed time intervals. Repeating timers can also have their next firing time manually adjusted.

A timer is not a real-time mechanism; it fires only when one of the run loop modes to which the timer has been added is running and able to check if the timer's firing time has passed. If a timer's firing time occurs while the run loop is in a mode that is not monitoring the timer or during a long callout, the timer does not fire until the next time the run loop checks the timer. Therefore, the actual time at which the timer fires potentially can be a significant period of time after the scheduled firing time.

A repeating timer reschedules itself based on the scheduled firing time, not the actual firing time. For example, if a timer is scheduled to fire at a particular time and every 5 seconds after that, the scheduled firing time will always fall on the original 5 second time intervals, even if the actual firing time gets delayed. If the firing time is delayed so far that it passes one or more of the scheduled firing times, the timer is fired only once for that time period; the timer is then rescheduled, after firing, for the next scheduled firing time in the future.

Each run loop timer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

CFRunLoopTimer is "toll-free bridged" with its Cocoa Foundation counterpart, NSTimer. 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 `NSTimer *` parameter, you can pass in a `CFRunLoopTimerRef`, and in a function where you see a `CFRunLoopTimerRef` parameter, you can pass in an `NSTimer` instance. This also applies to concrete subclasses of `NSTimer`. See Interchangeable Data Types for more information on toll-free bridging.

Functions

CFRunLoopTimerCreate

Creates a new CFRunLoopTimer object.

```
CFRunLoopTimerRef CFRunLoopTimerCreate (
   CFAllocatorRef allocator,
   CFAbsoluteTime fireDate,
   CFTimeInterval interval,
   CFOptionFlags flags,
   CFIndex order,
   CFRunLoopTimerCallBack callout,
   CFRunLoopTimerContext *context
);
```

Parameters

allocator

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

fireDate

The time at which the timer should first fire. The fine precision (sub-millisecond at most) of the fire date may be adjusted slightly by the timer if there are implementation reasons to do.

interval

The firing interval of the timer. If `0` or negative, the timer fires once and then is automatically invalidated. The fine precision (sub-millisecond at most) of the interval may be adjusted slightly by the timer if implementation reasons to do so exist.

flags

Currently ignored. Pass `0` for future compatibility.

order

A priority index indicating the order in which run loop timers are processed. Run loop timers currently ignore this parameter. Pass `0`.

callout

The callback function that is called when the timer fires.

context

A structure holding contextual information for the run loop timer. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be `NULL` if the callback function does not need the context's `info` pointer to keep track of state.

Return Value

The new CFRunLoopTimer object. Ownership follows the Create Rule.

Discussion

A timer needs to be added to a run loop mode before it will fire. To add the timer to a run loop, use `CFRunLoopAddTimer` (page 458). A timer can be registered to only one run loop at a time, although it can be in multiple modes within that run loop.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

GLUT

SampleUSBMIDIDriver

Worm

Declared In

`CFRunLoop.h`

CFRunLoopTimerDoesRepeat

Returns a Boolean value that indicates whether a CFRunLoopTimer object repeats.

```
Boolean CFRunLoopTimerDoesRepeat (
   CFRunLoopTimerRef timer
);
```

Parameters

timer

The run loop timer to test.

Return Value

`true` if *timer* repeats, or has a periodicity; otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopTimerGetContext

Returns the context information for a CFRunLoopTimer object.

```
void CFRunLoopTimerGetContext (
   CFRunLoopTimerRef timer,
   CFRunLoopTimerContext *context
);
```

Parameters

timer

The run loop timer to examine.

context

A pointer to the structure into which the context information for *timer* is to be copied. The information being returned is the same information passed to `CFRunLoopTimerCreate` (page 493) when creating *timer*.

Discussion

The context version number for run loop timers is currently 0. Before calling this function, you need to initialize the `version` member of *context* to `0`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopTimerGetInterval

Returns the firing interval of a repeating CFRunLoopTimer object.

```
CFTimeInterval CFRunLoopTimerGetInterval (
   CFRunLoopTimerRef timer
);
```

Parameters

timer

The run loop timer to examine.

Return Value

The firing interval of *timer*. Returns 0 if *timer* does not repeat.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Worm

Declared In

CFRunLoop.h

CFRunLoopTimerGetNextFireDate

Returns the next firing time for a CFRunLoopTimer object.

```
CFAbsoluteTime CFRunLoopTimerGetNextFireDate (
   CFRunLoopTimerRef timer
);
```

Parameters

timer

The run loop timer to examine.

Return Value

The next firing time for *timer*. This time could be a date in the past if a run loop has not been able to process the timer since the firing time arrived.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFRunLoop.h

CFRunLoopTimerGetOrder

Returns the ordering parameter for a CFRunLoopTimer object.

```
CFIndex CFRunLoopTimerGetOrder (
   CFRunLoopTimerRef timer
);
```

Parameters

timer

The run loop timer to examine.

Return Value
The ordering parameter for *timer*.

Discussion
The ordering parameter is currently ignored by run loop timers.

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

Declared In
CFRunLoop.h

CFRunLoopTimerGetTypeID

Returns the type identifier of the CFRunLoopTimer opaque type.

```
CFTypeID CFRunLoopTimerGetTypeID (
   void
);
```

Return Value
The type identifier for the CFRunLoopTimer opaque type.

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

Declared In
CFRunLoop.h

CFRunLoopTimerInvalidate

Invalidates a CFRunLoopTimer object, stopping it from ever firing again.

```
void CFRunLoopTimerInvalidate (
   CFRunLoopTimerRef timer
);
```

Parameters

timer
The run loop timer to invalidate.

Discussion
Once invalidated, *timer* will never fire and call its callback function again. This function automatically removes *timer* from all run loop modes in which it had been added. The memory is not deallocated unless the run loop held the only reference to *timer*.

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

Related Sample Code
GLUT
SampleUSBMIDIDriver
Worm

Declared In
`CFRunLoop.h`

CFRunLoopTimerIsValid

Returns a Boolean value that indicates whether a CFRunLoopTimer object is valid and able to fire.

```
Boolean CFRunLoopTimerIsValid (
   CFRunLoopTimerRef timer
);
```

Parameters

timer

The run loop timer to examine.

Return Value
`true` if *timer* is valid; otherwise `false`.

Discussion
A nonrepeating timer is automatically invalidated after it fires.

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

Declared In
`CFRunLoop.h`

CFRunLoopTimerSetNextFireDate

Sets the next firing date for a CFRunLoopTimer object .

```
void CFRunLoopTimerSetNextFireDate (
   CFRunLoopTimerRef timer,
   CFAbsoluteTime fireDate
);
```

Parameters

timer

The run loop timer to modify.

fireDate

The new firing time for *timer*.

Discussion
Resetting a timer's next firing time is a relatively expensive operation and should not be done if it can be avoided; letting timers autorepeat is more efficient. In some cases, however, manually-adjusted, repeating timers are useful. For example, if you have an action that will be performed multiple times in the future, but at irregular time intervals, it would be very expensive to create, add to run loop modes, and then destroy a timer for each firing event. Instead, you can create a repeating timer with an initial firing time in the distant future (or the initial firing time) and a very large repeat interval—on the order of decades or more—and add it to all the necessary run loop modes. Then, when you know when the timer should fire next, you reset the firing time with `CFRunLoopTimerSetNextFireDate`, perhaps from the timer's own callback function. This technique effectively produces a reusable, asynchronous timer.

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

Related Sample Code
GLUT
Worm

Declared In
`CFRunLoop.h`

Callbacks

CFRunLoopTimerCallBack

Callback invoked when a CFRunLoopTimer object fires.

```
typedef void (*CFRunLoopTimerCallBack) (
    CFRunLoopTimerRef timer,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFRunLoopTimerRef timer,
    void *info
);
```

Parameters

timer

The run loop timer that is firing.

info

The `info` member of the `CFRunLoopTimerContext` (page 500) structure that was used when creating the run loop timer.

Discussion
If *timer* repeats, the run loop automatically schedules the next firing time after calling this function, unless you manually update the firing time within this callback by calling `CFRunLoopTimerSetNextFireDate` (page 498). If *timer* does not repeat, the run loop invalidates *timer*.

You specify this callback when you create the timer with `CFRunLoopTimerCreate` (page 493).

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

Declared In
`CFRunLoop.h`

Data Types

CFRunLoopTimerContext

A structure that contains program-defined data and callbacks with which you can configure a CFRunLoopTimer's behavior.

```
struct CFRunLoopTimerContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFRunLoopTimerContext CFRunLoopTimerContext;
```

Fields

`version`
Version number of the structure. Must be 0.

`info`
An arbitrary pointer to program-defined data, which can be associated with the run loop timer at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

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

Declared In
`CFRunLoop.h`

CFRunLoopTimerRef

A reference to a run loop timer object.

```
typedef struct __CFRunLoopTimer *CFRunLoopTimerRef;
```

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

Declared In
`CFRunLoop.h`

CFSet Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFSet.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

CFSet and its derived mutable type, CFMutableSet, provide support for the mathematical concept of a set. A set, both in its mathematical sense and in the implementation of CFSet, is an unordered collection of distinct elements. CFSet creates static sets and CFMutableSet creates dynamic sets.

Use bags or sets as an alternative to arrays when the order of elements isn't important and performance in testing whether a value is contained in the collection is a consideration—while arrays are ordered, testing for membership is slower than with bags or sets. Use bags over sets if you want to allow duplicate values in your collections.

You create a static set object using either the `CFSetCreate` (page 504) or `CFSetCreateCopy` (page 505) function. These functions return a set containing the values you pass in as arguments. (Note that sets can't contain `NULL` pointers; in most cases, though, you can use the `kCFNull` constant instead.) Values are not copied but retained using the retain callback provided when the set was created. Similarly, when a value is removed from a set, it is released using the release callback.

CFSet provides functions for querying the values of a set. The `CFSetGetCount` (page 505) returns the number of values in a set, the `CFSetContainsValue` (page 503) function checks if a value is in a set, and `CFSetGetValues` (page 508) returns a C array containing all the values in a set.

CFSet is "toll-free bridged" with its Cocoa Foundation counterpart, NSSet. 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 `NSSet *` parameter, you can pass in a `CFSetRef`, and in a function where you see a `CFSetRef` parameter, you can pass in an NSSet instance. This also applies to concrete subclasses of NSSet. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating Sets

`CFSetCreate` (page 504)
Creates an immutable CFSet object containing supplied values.

`CFSetCreateCopy` (page 505)
Creates an immutable set containing the values of an existing set.

Examining a Set

`CFSetContainsValue` (page 503)
Returns a Boolean that indicates whether a set contains a given value.

`CFSetGetCount` (page 505)
Returns the number of values currently in a set.

`CFSetGetCountOfValue` (page 506)
Returns the number of values in a set that match a given value.

`CFSetGetValue` (page 507)
Obtains a specified value from a set.

`CFSetGetValueIfPresent` (page 507)
Reports whether or not a value is in a set, and if it exists returns the value indirectly.

`CFSetGetValues` (page 508)
Obtains all values in a set.

Applying a Function to Set Members

`CFSetApplyFunction` (page 502)
Calls a function once for each value in a set.

Getting the CFSet Type ID

`CFSetGetTypeID` (page 506)
Returns the type identifier for the CFSet type.

Functions

CFSetApplyFunction

Calls a function once for each value in a set.

```
void CFSetApplyFunction (
   CFSetRef theSet,
   CFSetApplierFunction applier,
   void *context
);
```

Parameters

theSet

The set to operate upon.

applier

The callback function to call once for each value in the *theSet*. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. The *applier* function must be able to work with all values in *theSet*.

context

A pointer-sized program-defined value, which is passed as the second parameter to the *applier* function, but is otherwise unused by this function.

Discussion

If *theSet* is mutable, it is unsafe for the *applier* function to change the contents of the collection.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

CFLocalServer

HID Calibrator

HID Config Save

HID Explorer

HID Utilities

Declared In

CFSet.h

CFSetContainsValue

Returns a Boolean that indicates whether a set contains a given value.

```
Boolean CFSetContainsValue (
   CFSetRef theSet,
   const void *value
);
```

Parameters

theSet

The set to search.

value

The value to match in *theSet*. Comparisons are made using the equal callback provided when *theSet* was created. If the equal callback was NULL, pointer equality (in C, ==) is used.

Return Value

true if *value* is contained in *theSet*, otherwise false.

Discussion
This function uses the equal callback. *value* and all elements in the set must be understood by the equal callback.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFLocalServer

Declared In
CFSet.h

CFSetCreate

Creates an immutable CFSet object containing supplied values.

```
CFSetRef CFSetCreate (
   CFAllocatorRef allocator,
   const void **values,
   CFIndex numValues,
   const CFSetCallBacks *callBacks
);
```

Parameters

allocator
The allocator to use to to allocate memory for the new set and its storage for values. Pass NULL or kCFAllocatorDefault to use the current default allocator.

values
A C array of the pointer-sized values to be in the new set. This parameter may be NULL if the *numValues* parameter is 0. The C array is not changed or freed by this function. values must be a pointer to a C array of at least numValues elements.

numValues
The number of values to copy from the values C array in the new set.

callBacks
A pointer to a CFSetCallBacks (page 512) structure initialized with the callbacks to use to retain, release, describe, and compare values in the collection. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in or can be reused for multiple collection creations.

This value may be NULL, which is treated as a valid structure of version 0 with all fields NULL. If the collection contains only CFType objects, then pass kCFTypeSetCallBacks (page 514) to use the default callback functions.

Return Value
A new immutable set, or NULL if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
If any value put into the collection is not one understood by one of the callback functions, the behavior when that callback function is used is undefined.

Availability
Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Declared In
CFSet.h

CFSetCreateCopy

Creates an immutable set containing the values of an existing set.

```
CFSetRef CFSetCreateCopy (
   CFAllocatorRef allocator,
   CFSetRef theSet
);
```

Parameters

allocator
The allocator to use to allocate memory for the new set and its storage for values. Pass NULL or kCFAllocatorDefault to use the current default allocator.

theSet
The set to copy.

Return Value
A new set that contains the same values as *theSet*, or NULL if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
The pointer values from *theSet* are copied into the new set, and the values are retained by the new set. The count of the new set is the same as the count of *theSet*. The new set uses the same callbacks as *theSet*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFSet.h

CFSetGetCount

Returns the number of values currently in a set.

```
CFIndex CFSetGetCount (
   CFSetRef theSet
);
```

Parameters

theSet
The set to examine.

Return Value
The number of values in *theSet*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFLocalServer
HID Dumper
HID LED test tool

Declared In
`CFSet.h`

CFSetGetCountOfValue

Returns the number of values in a set that match a given value.

```
CFIndex CFSetGetCountOfValue (
   CFSetRef theSet,
   const void *value
);
```

Parameters

`theSet`
The set to examine.

`value`
The value for which to search in *`theSet`*. Comparisons are made using the equal callback provided when *`theSet`* was created. If the equal callback was `NULL`, pointer equality (in C, ==) is used.

Return Value
The number of times *`value`* occurs in *`theSet`*. By definition, sets can not contain duplicate values, so returns `1` if *`value`* is contained in *`theSet`*, otherwise `0`.

Discussion
This function uses the equal callback. *`value`* and all elements in the set must be understood by the equal callback.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFSet.h`

CFSetGetTypeID

Returns the type identifier for the CFSet type.

```
CFTypeID CFSetGetTypeID (
   void
);
```

Return Value
The type identifier for the CFSet type.

Discussion
CFMutableSet has the same type identifier as CFSet.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFSet.h`

CFSetGetValue

Obtains a specified value from a set.

```
const void * CFSetGetValue (
   CFSetRef theSet,
   const void *value
);
```

Parameters

theSet
The set to examine.

value
The value for which to search in *theSet*. Comparisons are made using the equal callback provided when *theSet* was created. If the equal callback was `NULL`, pointer equality (in C, ==) is used.

Return Value
A pointer to the requested value, or `NULL` if the value is not in *theSet*. If the value is a Core Foundation object, Ownership follows the Get Rule.

Discussion
Since this function uses the equal callback, *value* all elements in the set must be understood by the equal callback. Depending on the implementation of the equal callback specified when creating *theSet*, the value returned may not have the same pointer equality as *value*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFSet.h`

CFSetGetValueIfPresent

Reports whether or not a value is in a set, and if it exists returns the value indirectly.

```
Boolean CFSetGetValueIfPresent (
   CFSetRef theSet,
   const void *candidate,
   const void **value
);
```

Parameters

theSet
The set to examine.

candidate

The value for which to search in *theSet*. Comparisons are made using the equal callback provided when *theSet* was created. If the equal callback was NULL, pointer equality (in C, ==) is used.

value

Upon return contains the matching value if it exists in *theSet*, otherwise NULL. If the value is a Core Foundation object, ownership follows the Get Rule.

Return Value
true if *value* exists in *theSet*, otherwise false.

Discussion
This function uses the equal callback. *candidate* and all elements in the set must be understood by the equal callback. Depending on the implementation of the equal callback specified when creating *theSet*, the value returned in *value* may not have the same pointer equality as *candidate*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFSet.h

CFSetGetValues

Obtains all values in a set.

```
void CFSetGetValues (
   CFSetRef theSet,
   const void **values
);
```

Parameters

theSet

The set to examine.

values

A C array of pointer-sized values to be filled with values from *theSet*. The value must be a valid C array of the appropriate type and of a size at least equal to the count of *theSet*). If the values are Core Foundation objects, ownership follows the Get Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFLocalServer
HID Dumper
HID LED test tool

Declared In
CFSet.h

Callbacks

CFSetApplierFunction

Prototype of a callback function that may be applied to every value in a set.

```
typedef void (*CFSetApplierFunction) (
   const void *value,
   void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   const void *value,
   void *context
);
```

Parameters

value

The current value in a set.

context

The program-defined context parameter given to the apply function.

Discussion

This callback is passed to the `CFSetApplyFunction` (page 502) function which iterates over the values in a set and applies the behavior defined in the applier function to each value in a set.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSet.h`

CFSetCopyDescriptionCallBack

Prototype of a callback function used to get a description of a value in a set.

```
typedef CFStringRef (*CFSetCopyDescriptionCallBack) (
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
   const void *value
);
```

Parameters

value

The value to be described.

Return Value
A textual description of *value*. The caller is responsible for releasing this object.

Discussion
This callback is passed to `CFSetCreate` (page 504) in a `CFSetCallBacks` (page 512) structure. This callback is used by the `CFCopyDescription` (page 652) function.

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

Declared In
`CFSet.h`

CFSetEqualCallBack

Prototype of a callback function used to determine if two values in a set are equal.

```
typedef Boolean (*CFSetEqualCallBack) (
   const void *value1,
   const void *value2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
   const void *value1,
   const void *value2
);
```

Parameters

value1
A value in the set.

value2
Another value in the set.

Return Value
`true` if *value1* and *value2* are equal, `false` otherwise.

Discussion
This callback is passed to `CFSetCreate` (page 504) in a `CFSetCallBacks` (page 512) structure.

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

Declared In
`CFSet.h`

CFSetHashCallBack

Prototype of a callback function called to compute a hash code for a value. Hash codes are used when values are accessed, added, or removed from a collection.

```
typedef CFHashCode     (*CFSetHashCallBack)
(
    const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFHashCode CFSetHashCallBack (
    const void *value
);
```

Parameters

value

The value used to compute the hash code.

Return Value

An integer that can be used as a table address in a hash table structure.

Discussion

This callback is passed to `CFSetCreate` (page 504) in a `CFSetCallBacks` (page 512) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSet.h`

CFSetReleaseCallBack

Prototype of a callback function used to release a value before it's removed from a set.

```
typedef void (*CFSetReleaseCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator

The set's allocator.

value

The value being removed from the set.

Discussion

This callback is passed to `CFSetCreate` (page 504) in a `CFSetCallBacks` (page 512) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In
CFSet.h

CFSetRetainCallBack

Prototype of a callback function used to retain a value being added to a set.

```
typedef const void *(*CFSetRetainCallBack) (
   CFAllocatorRef allocator,
   const void *value
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
   CFAllocatorRef allocator,
   const void *value
);
```

Parameters

allocator
The set's allocator.

value
The value being added to the set.

Return Value
The value to store in the set, which is usually the *value* parameter passed to this callback, but may be a different value if a different value should be stored in the collection.

Discussion
This callback is passed to `CFSetCreate` (page 504) in a `CFSetCallBacks` (page 512) structure.

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

Declared In
CFSet.h

Data Types

CFSetCallBacks

This structure contains the callbacks used to retain, release, describe, and compare the values of a CFSet object.

```
struct CFSetCallBacks {
    CFIndex version;
    CFSetRetainCallBack retain;
    CFSetReleaseCallBack release;
    CFSetCopyDescriptionCallBack copyDescription;
    CFSetEqualCallBack equal;
    CFSetHashCallBack hash;
};
typedef struct CFSetCallBacks CFSetCallBacks;
```

Fields

`version`

The version number of this structure. If not one of the defined version numbers for this opaque type, the behavior is undefined. The current version of this structure is `0`.

`retain`

The callback used to retain each value as they are added to the collection. If `NULL`, values are not retained. See `CFSetRetainCallBack` (page 512) for a descriptions of this function's parameters.

`release`

The callback used to release values as they are removed from the collection. If `NULL`, values are not released. See `CFSetReleaseCallBack` (page 511) for a description of this callback.

`copyDescription`

The callback used to create a descriptive string representation of each value in the collection. If `NULL`, the collection will create a simple description of each value. See `CFSetCopyDescriptionCallBack` (page 509) for a description of this callback.

`equal`

The callback used to compare values in the collection for equality for some operations. If `NULL`, the collection will use pointer equality to compare values in the collection. See `CFSetEqualCallBack` (page 510) for a description of this callback.

`hash`

The callback used to compute a hash code for values in a collection. If `NULL`, the collection computes a hash code by converting the pointer value to an integer. See `CFSetHashCallBack` (page 510) for a description of this callback.

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

Declared In
`CFSet.h`

CFSetRef

A reference to an immutable set object.

```
typedef const struct __CFSet *CFSetRef;
```

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

Declared In
`CFSet.h`

Constants

Predefined Callback Structures

CFSet provides some predefined callbacks for your convenience.

```
const CFSetCallBacks kCFTypeSetCallBacks;
const CFSetCallBacks kCFCopyStringSetCallBacks;
```

Constants

`kCFTypeSetCallBacks`

Predefined `CFSetCallBacks` (page 512) structure containing a set of callbacks appropriate for use when the values in a CFSet are all CFType-derived objects. The retain callback is `CFRetain` (page 657), the release callback is `CFRelease` (page 657), the copy callback is `CFCopyDescription` (page 652), the equal callback is `CFEqual` (page 653), and the hash callback is `CFHash` (page 656). Therefore, if you use this constant when creating the collection, items are automatically retained when added to the collection, and released when removed from the collection.

Available in Mac OS X v10.0 and later.

Declared in `CFSet.h`.

`kCFCopyStringSetCallBacks`

Predefined `CFSetCallBacks` (page 512) structure containing a set of callbacks appropriate for use when the values in a set are all CFString objects. The retain callback makes an immutable copy of strings added to the set.

Available in Mac OS X v10.0 and later.

Declared in `CFSet.h`.

CFSocket Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFSocket.h
Companion guides	CFNetwork Programming Guide Threading Programming Guide

Overview

A CFSocket is a communications channel implemented with a BSD socket.

CFSockets can be created from scratch (`CFSocketCreate` (page 518) and `CFSocketCreateWithSocketSignature` (page 522)), from a pre-existing BSD socket (`CFSocketCreateWithNative` (page 521)), or already connected to a remote socket (`CFSocketCreateConnectedToSocketSignature` (page 519)).

To listen for messages, you need to create a run loop source with `CFSocketCreateRunLoopSource` (page 520) and add it to a run loop with `CFRunLoopAddSource` (page 457). You can select the types of socket activities, such as connection attempts or data arrivals, that cause the source to fire and invoke your CFSocket's callback function. To send data, you store the data in a CFData and call `CFSocketSendData` (page 527).

Unlike Mach and message ports, sockets support communication over a network.

Functions by Task

Creating Sockets

`CFSocketCreate` (page 518)
Creates a CFSocket object of a specified protocol and type.

`CFSocketCreateConnectedToSocketSignature` (page 519)
Creates a CFSocket object and opens a connection to a remote socket.

`CFSocketCreateWithNative` (page 521)
Creates a CFSocket object for a pre-existing native socket.

`CFSocketCreateWithSocketSignature` (page 522)
Creates a CFSocket object using information from a CFSocketSignature structure.

Configuring Sockets

`CFSocketCopyAddress` (page 517)
Returns the local address of a CFSocket object.

`CFSocketCopyPeerAddress` (page 518)
Returns the remote address to which a CFSocket object is connected.

`CFSocketDisableCallBacks` (page 522)
Disables the callback function of a CFSocket object for certain types of socket activity.

`CFSocketEnableCallBacks` (page 523)
Enables the callback function of a CFSocket object for certain types of socket activity.

`CFSocketGetContext` (page 524)
Returns the context information for a CFSocket object.

`CFSocketGetNative` (page 524)
Returns the native socket associated with a CFSocket object.

`CFSocketGetSocketFlags` (page 525)
Returns flags that control certain behaviors of a CFSocket object.

`CFSocketSetAddress` (page 527)
Binds a local address to a CFSocket object.

`CFSocketSetSocketFlags` (page 528)
Sets flags that control certain behaviors of a CFSocket object.

Using Sockets

`CFSocketConnectToAddress` (page 516)
Opens a connection to a remote socket.

`CFSocketCreateRunLoopSource` (page 520)
Creates a CFRunLoopSource object for a CFSocket object.

`CFSocketGetTypeID` (page 525)
Returns the type identifier for the CFSocket opaque type.

`CFSocketInvalidate` (page 526)
Invalidates a CFSocket object, stopping it from sending or receiving any more messages.

`CFSocketIsValid` (page 526)
Returns a Boolean value that indicates whether a CFSocket object is valid and able to send or receive messages.

`CFSocketSendData` (page 527)
Sends data over a CFSocket object.

Functions

CFSocketConnectToAddress

Opens a connection to a remote socket.

```
CFSocketError CFSocketConnectToAddress (
   CFSocketRef s,
   CFDataRef address,
   CFTimeInterval timeout
);
```

Parameters

s

The CFSocket object with which to connect to *address*.

address

A CFData object containing a `struct sockaddr` appropriate for the protocol family of *s*, indicating the remote address to which to connect.

timeout

The time to wait for a connection to succeed. If a negative value is used, this function does not wait for the connection and instead lets the connection attempt happen in the background. If *s* requested a `kCFSocketConnectCallBack`, you will receive a callback when the background connection succeeds or fails.

Return Value
An error code indicating success or failure of the connection attempt.

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

Declared In
`CFSocket.h`

CFSocketCopyAddress

Returns the local address of a CFSocket object.

```
CFDataRef CFSocketCopyAddress (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value
The local address, stored as a `struct sockaddr` in a CFData object, of *s*. Ownership follows the Create Rule.

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

Related Sample Code
UDPEcho

Declared In
`CFSocket.h`

CFSocketCopyPeerAddress

Returns the remote address to which a CFSocket object is connected.

```
CFDataRef CFSocketCopyPeerAddress (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value

The remote address, stored as a `struct sockaddr` in a CFData object, to which *s* is connected. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

UDPEcho

Declared In

`CFSocket.h`

CFSocketCreate

Creates a CFSocket object of a specified protocol and type.

```
CFSocketRef CFSocketCreate (
   CFAllocatorRef allocator,
   SInt32 protocolFamily,
   SInt32 socketType,
   SInt32 protocol,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

allocator

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

protocolFamily

The protocol family for the socket. If negative or 0 is passed, the socket defaults to `PF_INET`.

socketType

The socket type to create. If *protocolFamily* is `PF_INET` and *socketType* is negative or 0, the socket type defaults to `SOCK_STREAM`.

protocol

The protocol for the socket. If *protocolFamily* is `PF_INET` and *protocol* is negative or 0, the socket protocol defaults to `IPPROTO_TCP` if *socketType* is `SOCK_STREAM` or `IPPROTO_UDP` if *socketType* is `SOCK_DGRAM`.

callBackTypes
A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 531) for the possible activity values.

callout
The function to call when one of the activities indicated by *callBackTypes* occurs.

context
A structure holding contextual information for the CFSocket object. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be NULL.

Return Value
The new CFSocket object, or NULL if an error occurred. Ownership follows the Create Rule.

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

Related Sample Code
CocoaEcho
CocoaHTTPServer
CocoaSOAP

Declared In
CFSocket.h

CFSocketCreateConnectedToSocketSignature

Creates a CFSocket object and opens a connection to a remote socket.

```
CFSocketRef CFSocketCreateConnectedToSocketSignature (
   CFAllocatorRef allocator,
   const CFSocketSignature *signature,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context,
   CFTimeInterval timeout
);
```

Parameters

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

signature
A CFSocketSignature (page 531) identifying the communication protocol and address to which the CFSocket object should connect.

callBackTypes
A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 531) for the possible activity values.

callout
The function to call when one of the activities indicated by *callBackTypes* occurs.

context

A structure holding contextual information for the CFSocket object. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be NULL.

timeout

The time to wait for a connection to succeed. If a negative value is used, this function does not wait for the connection and instead lets the connection attempt happen in the background. If *callBackTypes* includes kCFSocketConnectCallBack, you will receive a callback when the background connection succeeds or fails.

Return Value
The new CFSocket object, or NULL if an error occurred. Ownership follows the Create Rule.

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

Declared In
CFSocket.h

CFSocketCreateRunLoopSource

Creates a CFRunLoopSource object for a CFSocket object.

```
CFRunLoopSourceRef CFSocketCreateRunLoopSource (
   CFAllocatorRef allocator,
   CFSocketRef s,
   CFIndex order
);
```

Parameters

allocator

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

s

The CFSocket object for which to create a run loop source.

order

A priority index indicating the order in which run loop sources are processed. When multiple run loop sources are firing in a single pass through the run loop, the sources are processed in increasing order of this parameter. If the run loop is set to process only one source per loop, only the highest priority source, the one with the lowest *order* value, is processed.

Return Value
The new CFRunLoopSource object for *s*. Ownership follows the Create Rule.

Discussion
The run loop source is not automatically added to a run loop. To add the source to a run loop, use CFRunLoopAddSource (page 457).

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

Related Sample Code
CFLocalServer

CocoaEcho
CocoaHTTPServer
CocoaSOAP
UDPEcho

Declared In
`CFSocket.h`

CFSocketCreateWithNative

Creates a CFSocket object for a pre-existing native socket.

```
CFSocketRef CFSocketCreateWithNative (
   CFAllocatorRef allocator,
   CFSocketNativeHandle sock,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

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

sock
The native socket for which to create a CFSocket object.

callBackTypes
A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 531) for the possible activity values.

callout
The function to call when one of the activities indicated by *callBackTypes* occurs.

context
A structure holding contextual information for the CFSocket object. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be `NULL`.

Return Value
The new CFSocket object, or `NULL` if an error occurred. If a CFSocket object already exists for *sock*, the function returns the pre-existing object instead of creating a new object; the *context*, *callout*, and *callBackTypes* parameters are ignored in this case. Ownership follows the Create Rule.

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

Related Sample Code
CFLocalServer
DNSServiceMetaQuery
SimplePing
SRVResolver
UDPEcho

Declared In
CFSocket.h

CFSocketCreateWithSocketSignature

Creates a CFSocket object using information from a CFSocketSignature structure.

```
CFSocketRef CFSocketCreateWithSocketSignature (
   CFAllocatorRef allocator,
   const CFSocketSignature *signature,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

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

signature
A CFSocketSignature (page 531) identifying the communication protocol and address with which to create the CFSocket object.

callBackTypes
A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 531) for the possible activity values.

callout
The function to call when one of the activities indicated by *callBackTypes* occurs.

context
A structure holding contextual information for the CFSocket object. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call. Can be NULL.

Return Value
The new CFSocket object, or NULL if an error occurred. Ownership follows the Create Rule.

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

Declared In
CFSocket.h

CFSocketDisableCallBacks

Disables the callback function of a CFSocket object for certain types of socket activity.

```
void CFSocketDisableCallBacks (
   CFSocketRef s,
   CFOptionFlags callBackTypes
);
```

Parameters

s

The CFSocket object to modify.

callBackTypes

A bitwise-OR combination of CFSocket activity types that should not cause the callback function of *s* to be called. See Callback Types (page 531) for a list of callback types.

Discussion

If you no longer want certain types of callbacks that you requested when creating *s*, you can use this function to temporarily disable the callback. Use `CFSocketEnableCallBacks` (page 523) to reenable a callback type.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFSocket.h`

CFSocketEnableCallBacks

Enables the callback function of a CFSocket object for certain types of socket activity.

```
void CFSocketEnableCallBacks (
   CFSocketRef s,
   CFOptionFlags callBackTypes
);
```

Parameters

s

The CFSocket object to modify.

callBackTypes

A bitwise-OR combination of CFSocket activity types that should cause the callback function of *s* to be called. See Callback Types (page 531) for a list of callback types.

Discussion

If a callback type is not automatically reenabled, you can use this function to enable the callback. A callback type that is not automatically reenabled still does not get reenabled after enabling it with this function; use `CFSocketSetSocketFlags` (page 528) to have the callback type reenabled automatically.

Be sure to enable only callback types that your CFSocket object actually possesses and has requested when creating the CFSocket object; the result of enabling other callback types is undefined.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CFLocalServer

Declared In

`CFSocket.h`

CFSocketGetContext

Returns the context information for a CFSocket object.

```
void CFSocketGetContext (
   CFSocketRef s,
   CFSocketContext *context
);
```

Parameters

s

The CFSocket object to examine.

context

A pointer to the structure into which the context information for *s* is to be copied. The information being returned is usually the same information you passed to `CFSocketCreate` (page 518), `CFSocketCreateConnectedToSocketSignature` (page 519), `CFSocketCreateWithNative` (page 521), or `CFSocketCreateWithSocketSignature` (page 522) when creating the CFSocket object. However, if `CFSocketCreateWithNative` (page 521) returned a cached CFSocket object instead of creating a new object, *context* is filled with information from the original CFSocket object instead of the information you passed to the function.

Discussion

The context version number for CFSocket is currently 0. Before calling this function, you need to initialize the `version` member of *context* to 0.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketGetNative

Returns the native socket associated with a CFSocket object.

```
CFSocketNativeHandle CFSocketGetNative (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value

The native socket associated with *s*. If *s* has been invalidated, returns `-1`, `INVALID_SOCKET`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CocoaEcho

CocoaHTTPServer

CocoaSOAP

SimplePing

UDPEcho

Declared In
CFSocket.h

CFSocketGetSocketFlags

Returns flags that control certain behaviors of a CFSocket object.

```
CFOptionFlags CFSocketGetSocketFlags (
   CFSocketRef s
);
```

Parameters

s
The CFSocket to examine.

Return Value
A bitwise-OR combination of flags controlling the behavior of *s*. See CFSocket Flags (page 533) for the list of available flags.

Discussion
See `CFSocketSetSocketFlags` (page 528) for details on what the flags of a CFSocket mean.

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

Related Sample Code
DNSServiceMetaQuery
PortMapper
SimplePing
SRVResolver
UDPEcho

Declared In
CFSocket.h

CFSocketGetTypeID

Returns the type identifier for the CFSocket opaque type.

```
CFTypeID CFSocketGetTypeID ();
```

Return Value
The type identifier for the CFSocket opaque type.

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

Declared In
CFSocket.h

CFSocketInvalidate

Invalidates a CFSocket object, stopping it from sending or receiving any more messages.

```
void CFSocketInvalidate (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to invalidate.

Discussion

Invalidating a CFSocket object prevents the port from ever sending or receiving any more messages. The CFSocket object is not deallocated, though. The `CFSocketContext` (page 530) `info` information, which was provided when *s* was created, is released, if a release callback was specified in its context structure. Also, if a run loop source was created for *s*, the run loop source is invalidated, as well.

You should always invalidate a socket when you are done using it. If you have requested, using `CFSocketSetSocketFlags` (page 528), that the underlying socket not automatically close when invalidating the wrapping CFSocket object, you must invalidate the CFSocket object before closing the socket yourself.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFLocalServer
CocoaEcho
CocoaHTTPServer
CocoaSOAP
UDPEcho

Declared In

`CFSocket.h`

CFSocketIsValid

Returns a Boolean value that indicates whether a CFSocket object is valid and able to send or receive messages.

```
Boolean CFSocketIsValid (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value

`true` if *s* can be used for communication, otherwise `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketSendData

Sends data over a CFSocket object.

```
CFSocketError CFSocketSendData (
   CFSocketRef s,
   CFDataRef address,
   CFDataRef data,
   CFTimeInterval timeout
);
```

Parameters

s

The CFSocket object to use.

address

The address, stored as a `struct sockaddr` in a CFData object, to which to send the contents of *data*. If `NULL`, the data are sent to the address to which *s* is already connected.

data

The data to send.

timeout

The time to wait for the data to be sent.

Return Value

An error code indicating success or failure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketSetAddress

Binds a local address to a CFSocket object.

```
CFSocketError CFSocketSetAddress (
   CFSocketRef s,
   CFDataRef address
);
```

Parameters

s

The CFSocket object to modify.

address

A CFData object containing a `struct sockaddr` appropriate for the protocol family of *s*.

Return Value

An error code indicating success or failure.

Discussion

Once *s* is bound to *address*, depending on the socket's protocol, other processes and computers can connect to *s*.

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

Related Sample Code
CocoaEcho
CocoaHTTPServer
CocoaSOAP

Declared In
`CFSocket.h`

CFSocketSetSocketFlags

Sets flags that control certain behaviors of a CFSocket object.

```
void CFSocketSetSocketFlags (
   CFSocketRef s,
   CFOptionFlags flags
);
```

Parameters

s
The CFSocket object to modify.

flags
A bitwise-OR combination of flags controlling the behavior of *s*. See CFSocket Flags (page 533) for the list of available flags.

Discussion
The *flags* argument controls whether callbacks of a given type are automatically reenabled after they are triggered, and whether the underlying native socket is closed when *s* is invalidated.

By default `kCFSocketReadCallBack`, `kCFSocketAcceptCallBack`, and `kCFSocketDataCallBack` callbacks are automatically reenabled, whereas `kCFSocketWriteCallBack` callbacks are not; `kCFSocketConnectCallBack` callbacks can only occur once, so they cannot be reenabled. Be careful about automatically re-enabling read and write callbacks, because this implies that the callbacks will be sent repeatedly if the socket remains readable or writable respectively. Be sure to set these flags only for callback types that your CFSocket object actually possesses; the result of setting them for other callback types is undefined.

By default the underlying native socket will be closed when *s* is invalidated, but it will not be if the `kCFSocketCloseOnInvalidate` flag is turned off. This can be useful when you want to destroy a CFSocket object but continue to use the underlying native socket. The CFSocket object must still be invalidated when it will no longer be used. Do not in either case close the underlying native socket without invalidating the CFSocket object.

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

Related Sample Code
DNSServiceMetaQuery
PortMapper
SRVResolver

Declared In
CFSocket.h

Callbacks

CFSocketCallBack

Callback invoked when certain types of activity takes place on a CFSocket object.

```
typedef void (*CFSocketCallBack) (
    CFSocketRef s,
    CFSocketCallBackType callbackType,
    CFDataRef address,
    const void *data,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFSocketRef s,
    CFSocketCallBackType callbackType,
    CFDataRef address,
    const void *data,
    void *info
);
```

Parameters

s
The CFSocket object that experienced some activity.

callbackType
The type of activity detected.

address
A CFData object holding the contents of a `struct sockaddr` appropriate for the protocol family of *s*, identifying the remote address to which *s* is connected. This value is `NULL` except for `kCFSocketAcceptCallBack` and `kCFSocketDataCallBack` callbacks.

data
Data appropriate for the callback type. For a `kCFSocketConnectCallBack` that failed in the background, it is a pointer to an `SInt32` error code; for a `kCFSocketAcceptCallback`, it is a pointer to a `CFSocketNativeHandle` (page 530); or for a `kCFSocketDataCallBack`, it is a CFData object containing the incoming data. In all other cases, it is `NULL`.

info
The `info` member of the `CFSocketContext` (page 530) structure that was used when creating the CFSocket object.

Discussion

You specify this callback when you create the CFSocket object with `CFSocketCreate` (page 518), `CFSocketCreateConnectedToSocketSignature` (page 519), `CFSocketCreateWithNative` (page 521), or `CFSocketCreateWithSocketSignature` (page 522).

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

Declared In
`CFSocket.h`

Data Types

CFSocketContext

A structure that contains program-defined data and callbacks with which you can configure a CFSocket object's behavior.

```
struct CFSocketContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFSocketContext CFSocketContext;
```

Fields

`version`
Version number of the structure. Must be `0`.

`info`
An arbitrary pointer to program-defined data, which can be associated with the CFSocket object at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`
A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`
A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`
A copy description callback for your program-defined `info` pointer. Can be `NULL`.

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

Declared In
`CFSocket.h`

CFSocketNativeHandle

Type for the platform-specific native socket handle.

```
typedef int CFSocketNativeHandle;
```

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

Declared In
`CFSocket.h`

CFSocketRef

A reference to a CFSocket object.

```
typedef struct __CFSocket *CFSocketRef;
```

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

Declared In
`CFSocket.h`

CFSocketSignature

A structure that fully specifies the communication protocol and connection address of a CFSocket object.

```
struct CFSocketSignature {
    SInt32 protocolFamily;
    SInt32 socketType;
    SInt32 protocol;
    CFDataRef address;
};
typedef struct CFSocketSignature CFSocketSignature;
```

Fields

`protocolFamily`
The protocol family of the socket.

`socketType`
The socket type of the socket.

`protocol`
The protocol type of the socket.

`address`
A CFData object holding the contents of a `struct sockaddr` appropriate for the given protocol family, identifying the address of the socket.

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

Declared In
`CFSocket.h`

Constants

Callback Types

Types of socket activity that can cause the callback function of a CFSocket object to be called.

```
enum CFSocketCallBackType {
    kCFSocketNoCallBack = 0,
    kCFSocketReadCallBack = 1,
    kCFSocketAcceptCallBack = 2,
    kCFSocketDataCallBack = 3,
    kCFSocketConnectCallBack = 4,
    kCFSocketWriteCallBack = 8
};
typedef enum CFSocketCallBackType CFSocketCallBackType;
```

Constants

`kCFSocketNoCallBack`
No callback should be made for any activity.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketReadCallBack`
The callback is called when data is available to be read or a new connection is waiting to be accepted. The data is not automatically read; the callback must read the data itself.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketAcceptCallBack`
New connections will be automatically accepted and the callback is called with the data argument being a pointer to a `CFSocketNativeHandle` (page 530) of the child socket. This callback is usable only with listening sockets.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketDataCallBack`
Incoming data will be read in chunks in the background and the callback is called with the data argument being a CFData object containing the read data.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketConnectCallBack`
If a connection attempt is made in the background by calling `CFSocketConnectToAddress` (page 516) or `CFSocketCreateConnectedToSocketSignature` (page 519) with a negative timeout value, this callback type is made when the connect finishes. In this case the data argument is either `NULL` or a pointer to an `SInt32` error code, if the connect failed. This callback will never be sent more than once for a given socket.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketWriteCallBack`
The callback is called when the socket is writable. This callback type may be useful when large amounts of data are being sent rapidly over the socket and you want a notification when there is space in the kernel buffers for more data.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

Discussion
The callback types for which a callback is made is determined when the CFSocket object is created, such as with `CFSocketCreate` (page 518), or later with `CFSocketEnableCallBacks` (page 523) and `CFSocketDisableCallBacks` (page 522).

The `kCFSocketReadCallBack`, `kCFSocketAcceptCallBack`, and `kCFSocketDataCallBack` callbacks are mutually exclusive.

Version Notes
`kCFSocketWriteCallBack` is available in Mac OS X v10.2 and later.

CFSocket Flags

Flags that can be set on a CFSocket object to control its behavior.

```
enum {
    kCFSocketAutomaticallyReenableReadCallBack = 1,
    kCFSocketAutomaticallyReenableAcceptCallBack = 2,
    kCFSocketAutomaticallyReenableDataCallBack = 3,
    kCFSocketAutomaticallyReenableWriteCallBack = 8,
    kCFSocketCloseOnInvalidate = 128
};
```

Constants

`kCFSocketAutomaticallyReenableReadCallBack`
When enabled using `CFSocketSetSocketFlags` (page 528), the read callback is called every time the sockets has data to be read. When disabled, the read callback is called only once the next time data are available. The read callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableAcceptCallBack`
When enabled using `CFSocketSetSocketFlags` (page 528), the accept callback is called every time someone connects to your socket. When disabled, the accept callback is called only once the next time a new socket connection is accepted. The accept callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableDataCallBack`
When enabled using `CFSocketSetSocketFlags` (page 528), the data callback is called every time the socket has read some data. When disabled, the data callback is called only once the next time data are read. The data callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableWriteCallBack`
When enabled using `CFSocketSetSocketFlags` (page 528), the write callback is called every time more data can be written to the socket. When disabled, the write callback is called only the next time data can be written. The write callback is not automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketCloseOnInvalidate`
When enabled using `CFSocketSetSocketFlags` (page 528), the native socket associated with a CFSocket object is closed when the CFSocket object is invalidated. When disabled, the native socket remains open. This option is enabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

Discussion
The flags for a CFSocket object are set with `CFSocketSetSocketFlags` (page 528). To immediately enable or disable a callback, use `CFSocketEnableCallBacks` (page 523) and `CFSocketDisableCallBacks` (page 522).

Error Codes

Error codes for many CFSocket functions.

```
enum CFSocketError {
    kCFSocketSuccess = 0,
    kCFSocketError = -1,
    kCFSocketTimeout = -2
};
typedef enum CFSocketError CFSocketError;
```

Constants
`kCFSocketSuccess`
The socket operation succeeded.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketError`
The socket operation failed.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketTimeout`
The socket operation timed out.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

CFString Reference

Derived From:	*CFPropertyList Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFString.h CFBase.h
Companion guides	Property List Programming Topics for Core Foundation String 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 Cocoa and C-based programs. CFString objects are immutable—use `CFMutableStringRef` (page 358) to create and manage a string that can be changed after it has been created.

CFString has two primitive functions, `CFStringGetLength` (page 576) and `CFStringGetCharacterAtIndex` (page 569), 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, you can hash CFStrings using the `CFHash` (page 656) function. You should never, though, store a hash value outside of your application and expect it to be useful if you read it back in later (hash values may change between different releases of the operating system).

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 541)
Creates an immutable string from a constant compile-time string.

`CFStringCreateArrayBySeparatingStrings` (page 547)
Creates an array of CFString objects from a single CFString object.

`CFStringCreateByCombiningStrings` (page 549)
Creates a single string from the individual CFString objects that comprise the elements of an array.

`CFStringCreateCopy` (page 550)
Creates an immutable copy of a string.

`CFStringCreateFromExternalRepresentation` (page 552)
Creates a string from its "external representation."

`CFStringCreateWithBytes` (page 553)
Creates a string from a buffer containing characters in a specified encoding.

`CFStringCreateWithBytesNoCopy` (page 554)
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 555)
Creates a string from a buffer of Unicode characters.

`CFStringCreateWithCharactersNoCopy` (page 556)
Creates a string from a buffer of Unicode characters that might serve as the backing store for the object.

`CFStringCreateWithCString` (page 557)
Creates an immutable string from a C string.

`CFStringCreateWithCStringNoCopy` (page 558)
Creates a CFString object from an external C string buffer that might serve as the backing store for the object.

`CFStringCreateWithFormat` (page 559)
Creates an immutable string from a formatted string and a variable number of arguments.

`CFStringCreateWithFormatAndArguments` (page 560)
Creates an immutable string from a formatted string and a variable number of arguments (specified in a parameter of type `va_list`).

`CFStringCreateWithPascalString` (page 561)
Creates an immutable CFString object from a Pascal string.

`CFStringCreateWithPascalStringNoCopy` (page 562)
Creates a CFString object from an external Pascal string buffer that might serve as the backing store for the object.

`CFStringCreateWithSubstring` (page 563)
Creates an immutable string from a segment (substring) of an existing string.

Searching Strings

`CFStringCreateArrayWithFindResults` (page 548)
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 564)
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 565)
Query the range of the first character contained in the specified character set.

`CFStringFindWithOptions` (page 566)
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 567)
Returns a Boolean value that indicates whether a given string was found in a given source string.

`CFStringGetLineBounds` (page 577)
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 542)
Compares one string with another string.

`CFStringCompareWithOptions` (page 543)
Compares a range of the characters in one string with that of another string.

`CFStringCompareWithOptionsAndLocale` (page 544)
Compares a range of the characters in one string with another string using a given locale.

`CFStringHasPrefix` (page 586)
Determines if the character data of a string begin with a specified sequence of characters.

`CFStringHasSuffix` (page 587)
Determines if a string ends with a specified sequence of characters.

Accessing Characters

`CFStringCreateExternalRepresentation` (page 551)
Creates an “external representation” of a CFString object, that is, a CFData object.

`CFStringGetBytes` (page 568)
Fetches a range of the characters from a string into a byte buffer after converting the characters to a specified encoding.

`CFStringGetCharacterAtIndex` (page 569)
Returns the Unicode character at a specified location in a string.

`CFStringGetCharacters` (page 571)
Copies a range of the Unicode characters from a string to a user-provided buffer.

`CFStringGetCharactersPtr` (page 571)
Quickly obtains a pointer to the contents of a string as a buffer of Unicode characters.

`CFStringGetCharacterFromInlineBuffer` (page 570)
Returns the Unicode character at a specific location in an in-line buffer.

`CFStringGetCString` (page 572)
Copies the character contents of a string to a local C string buffer after converting the characters to a given encoding.

`CFStringGetCStringPtr` (page 573)
Quickly obtains a pointer to a C-string buffer containing the characters of a string in a given encoding.

`CFStringGetLength` (page 576)
Returns the number (in terms of UTF-16 code pairs) of Unicode characters in a string.

`CFStringGetPascalString` (page 582)
Copies the character contents of a CFString object to a local Pascal string buffer after converting the characters to a requested encoding.

`CFStringGetPascalStringPtr` (page 583)
Quickly obtains a pointer to a Pascal buffer containing the characters of a string in a given encoding.

`CFStringGetRangeOfComposedCharactersAtIndex` (page 583)
Returns the range of the composed character sequence at a specified index.

`CFStringInitInlineBuffer` (page 587)
Initializes an in-line buffer to use for efficient access of a CFString object's characters.

Working With Encodings

`CFStringConvertEncodingToIANACharSetName` (page 544)
Returns the name of the IANA registry "charset" that is the closest mapping to a specified string encoding.

`CFStringConvertEncodingToNSStringEncoding` (page 545)
Returns the Cocoa encoding constant that maps most closely to a given Core Foundation encoding constant.

`CFStringConvertEncodingToWindowsCodepage` (page 545)
Returns the Windows codepage identifier that maps most closely to a given Core Foundation encoding constant.

`CFStringConvertIANACharSetNameToEncoding` (page 546)
Returns the Core Foundation encoding constant that is the closest mapping to a given IANA registry "charset" name.

`CFStringConvertNSStringEncodingToEncoding` (page 546)
Returns the Core Foundation encoding constant that is the closest mapping to a given Cocoa encoding.

`CFStringConvertWindowsCodepageToEncoding` (page 547)
Returns the Core Foundation encoding constant that is the closest mapping to a given Windows codepage identifier.

`CFStringGetFastestEncoding` (page 575)
Returns for a CFString object the character encoding that requires the least conversion time.

`CFStringGetListOfAvailableEncodings` (page 578)
Returns a pointer to a list of string encodings supported by the current system.

`CFStringGetMaximumSizeForEncoding` (page 579)
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 580)
Returns the most compatible Mac OS script value for the given input encoding.

`CFStringGetNameOfEncoding` (page 580)
Returns the canonical name of a specified string encoding.

`CFStringGetSmallestEncoding` (page 584)
Returns the smallest encoding on the current system for the character contents of a string.

`CFStringGetSystemEncoding` (page 585)
Returns the default encoding used by the operating system when it creates strings.

`CFStringIsEncodingAvailable` (page 588)
Determines whether a given Core Foundation string encoding is available on the current system.

Getting Numeric Values

`CFStringGetDoubleValue` (page 574)
Returns the primary `double` value represented by a string.

`CFStringGetIntValue` (page 576)
Returns the integer value represented by a string.

Getting String Properties

`CFShowStr` (page 540)
Prints the attributes of a string during debugging.

`CFStringGetTypeID` (page 585)
Returns the type identifier for the CFString opaque type.

String File System Representations

`CFStringCreateWithFileSystemRepresentation` (page 559)
Creates a CFString from a zero-terminated POSIX file system representation.

`CFStringGetFileSystemRepresentation` (page 575)
Extracts the contents of a string as a `NULL`-terminated 8-bit string appropriate for passing to POSIX APIs.

`CFStringGetMaximumSizeOfFileSystemRepresentation` (page 579)
Determines the upper bound on the number of bytes required to hold the file system representation of the string.

Getting Paragraph Bounds

`CFStringGetParagraphBounds` (page 581)
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.

Managing Surrogates

`CFStringGetLongCharacterForSurrogatePair` (page 578)
Returns a UTF-32 character that corresponds to a given pair of UTF-16 surrogate characters.

`CFStringGetSurrogatePairForLongCharacter` (page 584)
Maps a given UTF-32 character to a pair of UTF-16 surrogate characters.

`CFStringIsSurrogateHighCharacter` (page 588)
Returns a Boolean value that indicates whether a given character is a high character in a surrogate pair.

`CFStringIsSurrogateLowCharacter` (page 589)
Returns a Boolean value that indicates whether a given character is a low character in a surrogate pair.

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` (page 658).

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.

- You can retain and release values returned from `CFSTR` 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).

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

Related Sample Code
HID Calibrator
HID Config Save
HID Explorer
HID Utilities
MoreSCF

Declared In
`CFString.h`

CFStringCompare

Compares one string with another string.

```
CFComparisonResult CFStringCompare (
   CFStringRef theString1,
   CFStringRef theString2,
   CFStringCompareFlags 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 591) for the available flags.

Return Value
A `Comparison Results` (page 802) 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
DockBrowser
FSMegaInfo
ImageClient
MoreOSL
MoreSCF

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,
   CFStringCompareFlags 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 542). 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 591) for the available flags.

Return Value
A `Comparison Results` (page 802) 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 542) 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,
   CFStringCompareFlags 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 591) 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` (page 250)). 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 `Comparison Results` (page 802) 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 546) 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.

```
unsigned long 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 546) function is complementary to this function.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
XMLBrowser

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 547) 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 595) constant if the name is not recognized.

Discussion

The `CFStringConvertEncodingToIANACharSetName` (page 544) function is complementary to this function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

XMLBrowser

Declared In

`CFString.h`

CFStringConvertNSStringEncodingToEncoding

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

```
CFStringEncoding CFStringConvertNSStringEncodingToEncoding (
   unsigned long 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 595) constant if the mapping is not known.

Discussion

The `CFStringConvertEncodingToNSStringEncoding` (page 545) 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 595) constant if the mapping is not known.

Discussion

The `CFStringConvertEncodingToWindowsCodepage` (page 545) 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` (page 35) to use the current default allocator.

theString

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

separatorString

The string 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 in *Memory Management Programming Guide for Core Foundation*.

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.

separatorString is treated as a complete unit. If you specify `XYZ` as the separator string, then if *theString* is `aXbYZcXYZe`, then the returned array contains `aXbYZc` and `e`.

See also `CFStringCreateByCombiningStrings` (page 549).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MoreSCF

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,
   CFStringCompareFlags compareOptions
);
```

Parameters

alloc

The allocator to use to allocate memory for the new CFArray object. Pass `NULL` or `kCFAllocatorDefault` (page 35) 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 591) for the available flags.

Return Value

An array that contains pointers to `CFRange` (page 802) 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 in *Memory Management Programming Guide for Core Foundation*.

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

Discussion
See also CFStringCreateArrayBySeparatingStrings (page 547).

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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 573). 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 Config Save
HID Explorer
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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 552) 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.

Related Sample Code
Dictionary
Preferences
String

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 551) function complements this function by creating an "external representation" CFData object from a string.

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

Related Sample Code
Dictionary
String

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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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
DisplayURL
ImageClient
MFSLives
MoreSCF
QTMetaData

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 (page 35) 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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 553)
`CFStringCreateWithCharactersNoCopy` (page 556)
`CFStringCreateWithCStringNoCopy` (page 558)
`CFStringCreateWithPascalStringNoCopy` (page 562)

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 571).

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

Related Sample Code
CoreTextTest
MoreSCF
QTSetMovieAudioDevice
RecentItems

SampleAUs

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 (page 35) 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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 555)
`CFStringCreateWithBytesNoCopy` (page 554)
`CFStringCreateWithCStringNoCopy` (page 558)
`CFStringCreateWithPascalStringNoCopy` (page 562)

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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
DiagnosticAUs
DockBrowser
HID Calibrator
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 (page 35) 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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 557)
CFStringCreateWithBytesNoCopy (page 554)
CFStringCreateWithCharactersNoCopy (page 556)
CFStringCreateWithPascalStringNoCopy (page 562)

Related Sample Code

String

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 (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

HID Calibrator

HID Config Save

HID Dumper

HID Explorer

HID Utilities

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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.

Related Sample Code

Dictionary

Preferences

SampleRaster

String

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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

FSMegaInfo

QTAudioContextInsert

QTKitTimeCode

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` (page 35) 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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

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 561)
`CFStringCreateWithBytesNoCopy` (page 554)
`CFStringCreateWithCStringNoCopy` (page 558)
`CFStringCreateWithCharactersNoCopy` (page 556)

Related Sample Code

String

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` (page 35) 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 in *Memory Management Programming Guide for Core Foundation*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioQueueTools

DockBrowser

MoreSCF

SpellingChecker CarbonCocoa Bundled

SpellingChecker-CocoaCarbon

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,
   CFStringCompareFlags 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 591) for the available flags.

Return Value
The range of the located substring within *theString*. If a match is not located, the returned CFRange (page 802) structure will have a location of kCFNotFound (page 803) 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 566) 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 548) 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

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,
   CFStringCompareFlags 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 591) and kCFCompareAnchored (page 592). 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,
   CFStringCompareFlags 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 591), kCFCompareAnchored (page 592), kCFCompareCaseInsensitive (page 591), kCFCompareNonliteral (page 592), and kCFCompareLocalized (page 592) (available in Mac OS X v10.0 and later). Uses the current user locale (the return value from CFLocaleCopyCurrent (page 242)) 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 564). 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 548) 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
MoreOSL
MoreSCF

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,
   CFStringCompareFlags 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 591) for possible values. kCFCompareNumerically (page 592) is ignored.

locale
The locale to use for the search comparison. NULL specifies the canonical locale (the return value from CFLocaleGetSystem (page 250)).

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 551) function also handles external representations and performs lossy conversions. The complementary function CFStringCreateWithBytes (page 553) 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
DisplayURL
MFSLives

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 571) function or the in-line buffer functions (`CFStringInitInlineBuffer` (page 587) and `CFStringGetCharacterFromInlineBuffer` (page 570)) as alternatives.

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

Related Sample Code
AuthForAll
DisplayURL
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 590) structure in which the characters are stored. You should initialize the structure with the `CFStringInitInlineBuffer` (page 587) 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 587) function. The in-line buffer functions, along with the `CFStringInlineBuffer` (page 590) 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.

Related Sample Code
String

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 587) and `CFStringGetCharacterFromInlineBuffer` (page 570) to extract the characters incrementally.

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

Related Sample Code
BSDLLCTest
EmbededAppleScripts
MFSLives
MoreSCF
QTMetaData

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 571) 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` (page 347)).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

String

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. On return, the buffer contains the converted characters. If there is an error in conversion, the buffer contains only partial results.

The buffer must be large enough to contain the converted characters and a `NUL` terminator. For example, if the string is `Toby`, the buffer must be at least 5 bytes long.

bufferSize
The length of *buffer* in bytes.

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 573) function fails.

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

Related Sample Code
GLUT
HID Calibrator
HID Config Save
HID Dumper
HID Utilities

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 572) 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
HID Manager Basics
InkSample

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.

Related Sample Code
CarbonCocoaTempConverter

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 579) if you want to make sure the buffer is of sufficient length.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

BetterAuthorizationSample

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

CFHostSample

HID Config Save

InkSample

MFSLives

MoreSCF

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 595)-terminated list of enum constants, each of type CFStringEncoding (page 589).

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit

Declared In
CFString.h

CFStringGetLongCharacterForSurrogatePair

Returns a UTF-32 character that corresponds to a given pair of UTF-16 surrogate characters.

```
UTF32Char CFStringGetLongCharacterForSurrogatePair (
   UniChar surrogateHigh,
   UniChar surrogateLow
);
```

Parameters

surrogateHigh
The high surrogate character.

surrogateLow
The low surrogate character.

Return Value
A UTF32Char that corresponds to the combination of *surrogateHigh* and *surrogateLow*.

Discussion

Availability
Available in Mac OS X v10.6 and later.

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

BetterAuthorizationSample

CFHostSample

FSMegaInfo

IOPrintSuperClasses

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 575).

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

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 in *Memory Management Programming Guide for Core Foundation*.

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 577)(), however it only looks 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 583) function fails.

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

Related Sample Code
FSMegaInfo
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 582) 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.

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`

CFStringGetSurrogatePairForLongCharacter

Maps a given UTF-32 character to a pair of UTF-16 surrogate characters.

```
Boolean CFStringGetSurrogatePairForLongCharacter (
   UTF32Char character,
   UniChar *surrogates
);
```

Parameters

character
A UTF-32 character.

surrogates
A buffer to contion the returned surrogate pair.

The buffer must have space for at least 2 UTF-16 characters.

Return Value
`true` if *character* is mapped to a surrogate pair, otherwise `false`.

Discussion

Availability
Available in Mac OS X v10.6 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
BasicInputMethod
BSDLLCTest
GLUT
HID Manager Basics
HID Utilities Source

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
BetterAuthorizationSample
CFFTPSample
FSMegaInfo
ImageApp
MoreSCF

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
HID Calibrator
HID Config Save
HID Explorer
MoreSCF

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

MoreSCF

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 590) structure to initialize. On return, an initialized structure that can be used in a `CFStringGetCharacterFromInlineBuffer` (page 570) 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 590) structure that can be used for accessing the characters of a string. Once the buffer is initialized you can call the `CFStringGetCharacterFromInlineBuffer` (page 570) function to access the characters in the buffer one at a time. The in-line buffer functions, along with the `CFStringInlineBuffer` (page 590) 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.

Related Sample Code
String

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`

CFStringIsSurrogateHighCharacter

Returns a Boolean value that indicates whether a given character is a high character in a surrogate pair.

```
Boolean CFStringIsSurrogateHighCharacter (
   UniChar character
);
```

Parameters

character
A UTF-16 character.

Return Value
`true` if *character* is a high character in a surrogate pair, otherwise `false`.

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFString.h`

CFStringIsSurrogateLowCharacter

Returns a Boolean value that indicates whether a given character is a low character in a surrogate pair.

```
Boolean CFStringIsSurrogateLowCharacter (
   UniChar character
);
```

Parameters

character
A UTF-16 character.

Return Value
`true` if *character* is a low character in a surrogate pair, otherwise `false`.

Discussion

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFString.h`

Data Types

CFStringCompareFlags

A `CFOptionFlags` (page 801) type for specifying options for string comparison .

```
typedef CFOptionFlags CFStringCompareFlags;
```

Discussion
See "String Comparison Flags" (page 591) 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 593) for a list) and for platform-dependent encodings (see "External String Encodings" (page 595)). 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 595).

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 587) function for initializing the fields of this structure; do not do it manually. Once the buffer is initialized, use the `CFStringGetCharacterFromInlineBuffer` (page 570) 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 550) (instead of `CFRetain` (page 657)) 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 10".

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 542). 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 = 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,
    kCFStringEncodingUTF7 = 0x04000100,
    kCFStringEncodingUTF7_IMAP = 0x0A10,
    kCFStringEncodingShiftJIS_X0213_00 = 0x0628 /* Deprecated */
};
```

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

Available in Mac OS X v10.5 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`.

`kCFStringEncodingUTF7`
kTextEncodingUnicodeDefault + kUnicodeUTF7Format RFC2152.

Available in Mac OS X v10.6 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingUTF7_IMAP`
UTF-7 (IMAP folder variant) RFC3501.

Available in Mac OS X v10.6 and later.

Declared in `CFStringEncodingExt.h`.

`kCFStringEncodingShiftJIS_X0213_00`

Shift-JIS format encoding of JIS X0213 planes 1 and 2. (Deprecated. Deprecated. Use `kCFStringEncodingShiftJIS_X0213` (page 606) instead.)

Available in Mac OS X v10.2 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.

CFStringTokenizer Reference

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

Overview

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

You can use a CFStringTokenizer to break a string into tokens (sub-strings) on the basis of words, sentences, or paragraphs. When you create a tokenizer, you can supply options to further modify the tokenization—see "Tokenization Modifiers" (page 619).

In addition, with CFStringTokenizer:

- You can de-compound German compounds
- You can identify the language used in a string (using `CFStringTokenizerCopyBestStringLanguage` (page 614))
- You can obtain Latin transcription for tokens

To find a token that includes the character specified by character index and set it as the current token, you call `CFStringTokenizerGoToTokenAtIndex` (page 617). To advance to the next token and set it as the current token, you call `CFStringTokenizerAdvanceToNextToken` (page 613). To get the range of current token, you call `CFStringTokenizerGetCurrentTokenRange` (page 617). You can use `CFStringTokenizerCopyCurrentTokenAttribute` (page 614) to get the attribute of the current token. If the current token is a compound, you can call `CFStringTokenizerGetCurrentSubTokens` (page 616) to retrieve the subtokens or derived subtokens contained in the compound token. To guess the language of a string, you call `CFStringTokenizerCopyBestStringLanguage` (page 614).

CFStringTokenizer replaces the Language Analysis Manager (see *Language Analysis Manager Reference*). The Language Analysis Manager API provides access to one specific language engine at a time. For example you can create an analysis environment for Japanese tokenization, but it can't then be used to tokenize Chinese. Such API is good when you develop a language specific applications that handle a specific language such as input methods. It is not, however, convenient when you develop an internationalized applications which

handle text in language neutral way. Conceptually, CFStringTokenizer provides a higher level API that supports typical tasks of internationalized applications. With CFStringTokenizer you can tokenize a string without knowing the language.

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

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

Functions by Task

Creating a Tokenizer

`CFStringTokenizerCreate` (page 615)
Returns a tokenizer for a given string.

Setting the String

`CFStringTokenizerSetString` (page 618)
Sets the string for a tokenizer.

Changing the Location

`CFStringTokenizerAdvanceToNextToken` (page 613)
Advances the tokenizer to the next token and sets that as the current token.

`CFStringTokenizerGoToTokenAtIndex` (page 617)
Finds a token that includes the character at a given index, and set it as the current token.

Getting Information About the Current Token

`CFStringTokenizerCopyCurrentTokenAttribute` (page 614)
Returns a given attribute of the current token.

`CFStringTokenizerGetCurrentTokenRange` (page 617)
Returns the range of the current token.

`CFStringTokenizerGetCurrentSubTokens` (page 616)
Retrieves the subtokens or derived subtokens contained in the compound token.

Identifying a Language

`CFStringTokenizerCopyBestStringLanguage` (page 614)
Guesses a language of a given string and returns the guess as a BCP 47 string.

Getting the CFStringTokenizer Type ID

`CFStringTokenizerGetTypeID` (page 617)
Returns the type ID for CFStringTokenizer.

Functions

CFStringTokenizerAdvanceToNextToken

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

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

Parameters

tokenizer
A CFStringTokenizer object.

Return Value
The type of the token if the tokenizer succeeded in finding a token and setting it as current token. Returns `kCFStringTokenizerTokenNone` if the tokenizer failed to find a token. For possible values, see "CFStringTokenizerTokenType" (page 620).

Discussion
If there is no preceding call to `CFStringTokenizerGoToTokenAtIndex` (page 617) or `CFStringTokenizerAdvanceToNextToken`, the function finds the first token in the range specified by the `CFStringTokenizerCreate` (page 615). If there is a preceding, successful, call to `CFStringTokenizerGoToTokenAtIndex` (page 617) or `CFStringTokenizerAdvanceToNextToken` and there is a current token, proceeds to the next token. If a token is found, it is set as the current token and the function returns `true`; otherwise the current token is invalidates and the function returns `false`.

You can obtain the range and attribute of the token calling `CFStringTokenizerGetCurrentTokenRange` (page 617) and `CFStringTokenizerCopyCurrentTokenAttribute` (page 614). If the token is a compound (with type `kCFStringTokenizerTokenHasSubTokensMask` or `kCFStringTokenizerTokenHasDerivedSubTokensMask`), you can obtain its subtokens and (or) derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 616).

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

See Also
`CFStringTokenizerGoToTokenAtIndex` (page 617)

Declared In
CFStringTokenizer.h

CFStringTokenizerCopyBestStringLanguage

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

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

Parameters

string

The string to test to identify the language.

range

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

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

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

CRStringTokenizer recognizes the following languages:

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

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

Declared In
CFStringTokenizer.h

CFStringTokenizerCopyCurrentTokenAttribute

Returns a given attribute of the current token.

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

Parameters

tokenizer

A CFStringTokenizer object.

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

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

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

Declared In
`CFStringTokenizer.h`

CFStringTokenizerCreate

Returns a tokenizer for a given string.

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

Parameters

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

string
The string to tokenize.

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

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

For possible values, see "Tokenization Modifiers" (page 619).

locale
A locale that specifies language- or region-specific behavior for the tokenization. You can pass `NULL` to use the default system locale, although this is typically not recommended—instead use `CFLocaleCopyCurrent` (page 242) to specify the locale of the current user.

For more information, see "Tokenization Modifiers" (page 619).

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

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

Declared In
CFStringTokenizer.h

CFStringTokenizerGetCurrentSubTokens

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

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

Parameters

tokenizer
A CFStringTokenizer object.

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

maxRangeLength
The maximum number of ranges to return.

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

Return Value
The number of ranges returned.

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

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

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

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

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

Declared In
CFStringTokenizer.h

CFStringTokenizerGetCurrentTokenRange

Returns the range of the current token.

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

Parameters

tokenizer
A CFStringTokenizer object.

Return Value

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

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStringTokenizer.h`

CFStringTokenizerGetTypeID

Returns the type ID for CFStringTokenizer.

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

Return Value

The type ID for CFStringTokenizer.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStringTokenizer.h`

CFStringTokenizerGoToTokenAtIndex

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

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

Parameters

tokenizer
A CFStringTokenizer object.

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

Return Value
The type of the token if the tokenizer succeeded in finding a token and setting it as the current token. Returns `kCFStringTokenizerTokenNone` if the tokenizer failed to find a token. For possible values, see "CFStringTokenizerTokenType" (page 620).

Discussion
You can obtain the range and attribute of the token calling `CFStringTokenizerGetCurrentTokenRange` (page 617) and `CFStringTokenizerCopyCurrentTokenAttribute` (page 614). If the token is a compound (with type `kCFStringTokenizerTokenHasSubTokensMask` or `kCFStringTokenizerTokenHasDerivedSubTokensMask`), you can obtain its subtokens and (or) derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 616).

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

See Also
`CFStringTokenizerAdvanceToNextToken` (page 613)

Declared In
`CFStringTokenizer.h`

CFStringTokenizerSetString

Sets the string for a tokenizer.

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

Parameters

tokenizer
A tokenizer.

string
The string for the tokenizer to tokenize.

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

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

Declared In
`CFStringTokenizer.h`

Data Types

CFStringTokenizerRef

A reference to a CFStringTokenizer object.

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

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

Declared In
`CFStringTokenizer.h`

Constants

Tokenization Modifiers

Tokenization options are used with `CFStringTokenizerCreate` (page 615) to specify how the string should be tokenized

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

Constants

`kCFStringTokenizerUnitWord`
Specifies that a string should be tokenized by word. The *`locale`* parameter of `CFStringTokenizerCreate` (page 615) is ignored.
Available in Mac OS X v10.5 and later.
Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitSentence`
Specifies that a string should be tokenized by sentence. The *`locale`* parameter of `CFStringTokenizerCreate` (page 615) is ignored.
Available in Mac OS X v10.5 and later.
Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitParagraph`
Specifies that a string should be tokenized by paragraph. The *`locale`* parameter of `CFStringTokenizerCreate` (page 615) is ignored.
Available in Mac OS X v10.5 and later.
Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerUnitLineBreak`
Specifies that a string should be tokenized by line break. The *`locale`* parameter of `CFStringTokenizerCreate` (page 615) is ignored.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

You can use this constant in double-click range detection and whole word search. It is locale-sensitive. If the locale is `en_US_POSIX`, a colon (U+003A) is treated as a word separator. If the *`locale`* parameter of `CFStringTokenizerCreate` (page 615) is `NULL`, the locale from the global `AppleTextBreakLocale` preference is used if it is available; otherwise the locale defaults to the first locale in `AppleLanguages`.

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

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

Used with `kCFStringTokenizerUnitSentence` (page 619) or `kCFStringTokenizerUnitParagraph` (page 619).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

Discussion
You use the tokenization unit options with `CFStringTokenizerCreate` (page 615) to specify how a string should be tokenized.

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

You use the attribute specifiers to tell the tokenizer to prepare the specified attribute when it tokenizes the given string. You can retrieve the attribute value by calling `CFStringTokenizerCopyCurrentTokenAttribute` (page 614) with one of the attribute options.

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

CFStringTokenizerTokenType

Token types returned by `CFStringTokenizerGoToTokenAtIndex` (page 617) and `CFStringTokenizerAdvanceToNextToken` (page 613).

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

Constants

`kCFStringTokenizerTokenNone`
Has no token.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenNormal`
Has a normal token.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

You can obtain subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 616).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasDerivedSubTokensMask`
Compound token which may contain derived subtokens.

You can obtain subtokens and derived subtokens by calling `CFStringTokenizerGetCurrentSubTokens` (page 616).

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenHasHasNumbersMask`
Appears to contain a number.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

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

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

`kCFStringTokenizerTokenIsCJWordMask`
Contains kana and/or ideographs.

Available in Mac OS X v10.5 and later.

Declared in `CFStringTokenizer.h`.

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

CFTimeZone Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFTimeZone.h CFDate.h
Companion guide	Date and Time Programming Guide for Core Foundation

Overview

CFTimeZone defines the behavior of time zone objects. Time zone objects represent geopolitical regions. Consequently, these objects have names for these regions. Time zone objects also represent a temporal offset, either plus or minus, from Greenwich Mean Time (GMT) and an abbreviation (such as PST for Pacific Standard Time).

CFTimeZone provides several functions to create time zone objects: `CFTimeZoneCreateWithName` (page 628) and `CFTimeZoneCreateWithTimeIntervalFromGMT` (page 629). CFTimeZone also permits you to set the default time zone within your application using the `CFTimeZoneSetDefault` (page 633) function. You can access this default time zone at any time with the `CFTimeZoneCopyDefault` (page 626) function.

CFTimeZone is "toll-free bridged" with its Cocoa Foundation counterpart, NSTimeZone. 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 `NSTimeZone *` parameter, you can pass in a `CFTimeZoneRef`, and in a function where you see a `CFTimeZoneRef` parameter, you can pass in an NSTimeZone instance. This fact also applies to concrete subclasses of NSTimeZone. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a Time Zone

`CFTimeZoneCreateWithName` (page 628)
Returns the time zone object identified by a given name or abbreviation.

`CFTimeZoneCreateWithTimeIntervalFromGMT` (page 629)
Returns a time zone object for the specified time interval offset from Greenwich Mean Time (GMT).

`CFTimeZoneCreate` (page 627)
Creates a time zone with a given name and data.

System and Default Time Zones and Information

`CFTimeZoneCopyAbbreviationDictionary` (page 625)
Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

`CFTimeZoneCopyAbbreviation` (page 625)
Returns the abbreviation of a time zone at a specified date.

`CFTimeZoneCopyDefault` (page 626)
Returns the default time zone set for your application.

`CFTimeZoneCopySystem` (page 627)
Returns the time zone currently used by the system.

`CFTimeZoneSetDefault` (page 633)
Sets the default time zone for your application the given time zone.

`CFTimeZoneCopyKnownNames` (page 626)
Returns an array of strings containing the names of all the time zones known to the system.

`CFTimeZoneResetSystem` (page 632)
Clears the previously determined system time zone, if any.

`CFTimeZoneSetAbbreviationDictionary` (page 633)
Sets the abbreviation dictionary to a given dictionary.

Getting Information About Time Zones

`CFTimeZoneGetName` (page 630)
Returns the geopolitical region name that identifies a given time zone.

`CFTimeZoneCopyLocalizedName` (page 626)
Returns the localized name of a given time zone.

`CFTimeZoneGetSecondsFromGMT` (page 631)
Returns the difference in seconds between the receiver and Greenwich Mean Time (GMT) at the specified date.

`CFTimeZoneGetData` (page 629)
Returns the data that stores the information used by a time zone.

Getting Daylight Savings Time Information

`CFTimeZoneIsDaylightSavingTime` (page 632)
Returns whether or not a time zone is in daylight savings time at a specified date.

`CFTimeZoneGetDaylightSavingTimeOffset` (page 630)
Returns the daylight saving time offset for a time zone at a given time.

`CFTimeZoneGetNextDaylightSavingTimeTransition` (page 630)
Returns the time in a given time zone of the next daylight saving time transition after a given time.

Getting the CFTimeZone Type ID

`CFTimeZoneGetTypeID` (page 631)
Returns the type identifier for the CFTimeZone opaque type.

Functions

CFTimeZoneCopyAbbreviation

Returns the abbreviation of a time zone at a specified date.

```
CFStringRef CFTimeZoneCopyAbbreviation (
   CFTimeZoneRef tz,
   CFAbsoluteTime at
);
```

Parameters

tz
The time zone to use.

at
The absolute time at which to obtain the abbreviation.

Return Value
A string containing the time zone abbreviation of *at*. Ownership follows the Create Rule.

Discussion
Note that the abbreviation may be different at different dates. For example, during daylight savings time the US/Eastern time zone has an abbreviation of "EDT." At other times, its abbreviation is "EST."

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

Declared In
`CFTimeZone.h`

CFTimeZoneCopyAbbreviationDictionary

Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

```
CFDictionaryRef CFTimeZoneCopyAbbreviationDictionary (
   void
);
```

Return Value
A dictionary containing the mappings of time zone abbreviations to time zone names. Ownership follows the Create Rule.

Discussion
More than one time zone may have the same abbreviation. For example, US/Pacific and Canada/Pacific both use the abbreviation "PST." In these cases this function chooses a single name to map the abbreviation to.

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

Declared In
CFTimeZone.h

CFTimeZoneCopyDefault

Returns the default time zone set for your application.

```
CFTimeZoneRef CFTimeZoneCopyDefault (
   void
);
```

Return Value
A time zone representing the default time zone set for your application, or the system time zone if no default is set. Ownership follows the Create Rule.

Discussion
If no default time zone is set, this function simply returns the result of the `CFTimeZoneCopySystem` (page 627) function.

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

Related Sample Code
MFSLives

Declared In
CFTimeZone.h

CFTimeZoneCopyKnownNames

Returns an array of strings containing the names of all the time zones known to the system.

```
CFArrayRef CFTimeZoneCopyKnownNames (
   void
);
```

Return Value
An array containing CFString objects representing all the known time zone names. Ownership follows the Create Rule.

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

Declared In
CFTimeZone.h

CFTimeZoneCopyLocalizedName

Returns the localized name of a given time zone.

```
CFStringRef CFTimeZoneCopyLocalizedName (
    CFTimeZoneRef tz,
    CFTimeZoneNameStyle style,
    CFLocaleRef locale
);
```

Parameters

tz

The time zone to analyze.

style

The style for the returned name.

locale

The locale for which to localize the returned name.

Return Value

The name of *tz* localized for *locale*. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

CFTimeZone.h

CFTimeZoneCopySystem

Returns the time zone currently used by the system.

```
CFTimeZoneRef CFTimeZoneCopySystem (
   void
);
```

Return Value

A time zone representing the time zone currently used by the system, or the GMT time zone if the current zone cannot be determined. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SampleDS

Declared In

CFTimeZone.h

CFTimeZoneCreate

Creates a time zone with a given name and data.

```
CFTimeZoneRef CFTimeZoneCreate (
   CFAllocatorRef allocator,
   CFStringRef name,
   CFDataRef data
);
```

Parameters

allocator

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

name

The name of the time zone to create.

data

The data to use to initialize the time zone. The contents of the data should be the same as that found within the time-zone files located at `/usr/share/zoneinfo`.

Return Value

A time zone corresponding to *name* and *data*. Ownership follows the Create Rule.

Discussion

You typically do not call this function directly. Use the `CFTimeZoneCreateWithName` (page 628) function to obtain a time zone given its name.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTimeZone.h`

CFTimeZoneCreateWithName

Returns the time zone object identified by a given name or abbreviation.

```
CFTimeZoneRef CFTimeZoneCreateWithName (
   CFAllocatorRef allocator,
   CFStringRef name,
   Boolean tryAbbrev
);
```

Parameters

allocator

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

name

The name or abbreviation of the time zone to obtain. The name may be in any of the formats understood by the system, for example "EST", "Etc/GMT-2", "America/Argentina/Buenos_Aires", "Europe/Monaco", "US/Pacific", or "posixrules". For a complete list of system names, you can see the output of CFTimeZoneCopyKnownNames (page 626).

tryAbbrev

If `false`, assumes *name* is not an abbreviation and searches the time zone information directory for a matching name. If `true`, tries to resolve *name* using the abbreviation dictionary first before searching the information dictionary.

Return Value
A time zone corresponding to *name*, or `NULL` if no match was found. Ownership follows the Create Rule.

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

Declared In
`CFTimeZone.h`

CFTimeZoneCreateWithTimeIntervalFromGMT

Returns a time zone object for the specified time interval offset from Greenwich Mean Time (GMT).

```
CFTimeZoneRef CFTimeZoneCreateWithTimeIntervalFromGMT (
   CFAllocatorRef allocator,
   CFTimeInterval ti
);
```

Parameters

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

ti
The offset, from GMT, of the new time zone.

Return Value
A new time zone whose offset from GMT is given by the interval *ti*. The name of the new time zone is GMT +/- the offset, in hours and minutes. Time zones created with this function never have daylight savings, and the offset is constant no matter what the date. Ownership follows the Create Rule.

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

Declared In
`CFTimeZone.h`

CFTimeZoneGetData

Returns the data that stores the information used by a time zone.

```
CFDataRef CFTimeZoneGetData (
   CFTimeZoneRef tz
);
```

Parameters

tz
The time zone to analyze.

Return Value
The data used to store *tz*. Ownership follows the Get Rule.

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

Declared In
CFTimeZone.h

CFTimeZoneGetDaylightSavingTimeOffset

Returns the daylight saving time offset for a time zone at a given time.

```
CFTimeInterval CFTimeZoneGetDaylightSavingTimeOffset (
   CFTimeZoneRef tz,
   CFAbsoluteTime at
);
```

Parameters

tz

The time zone to analyze.

at

The time in *tz* to test for daylight saving time offset.

Return Value
The daylight saving time offset for *tz* at *at*.

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

Declared In
CFTimeZone.h

CFTimeZoneGetName

Returns the geopolitical region name that identifies a given time zone.

```
CFStringRef CFTimeZoneGetName (
   CFTimeZoneRef tz
);
```

Parameters

tz

The time zone to analyze.

Return Value
A string containing the geopolitical region name that identifies *tz*. Ownership follows the Get Rule.

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

Declared In
CFTimeZone.h

CFTimeZoneGetNextDaylightSavingTimeTransition

Returns the time in a given time zone of the next daylight saving time transition after a given time.

```
CFAbsoluteTime CFTimeZoneGetNextDaylightSavingTimeTransition (
   CFTimeZoneRef tz,
   CFAbsoluteTime at
);
```

Parameters

tz

The time zone to analyze.

at

A time in *tz*.

Return Value
The time in *tz* of the next daylight saving time transition after *at*.

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

Declared In
CFTimeZone.h

CFTimeZoneGetSecondsFromGMT

Returns the difference in seconds between the receiver and Greenwich Mean Time (GMT) at the specified date.

```
CFTimeInterval CFTimeZoneGetSecondsFromGMT (
   CFTimeZoneRef tz,
   CFAbsoluteTime at
);
```

Parameters

tz

The time zone to analyze.

at

The date at which the interval is to be computed.

Return Value
The difference in seconds between *tz* and GMT at the specified date, *at*.

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

Related Sample Code
MFSLives

Declared In
CFTimeZone.h

CFTimeZoneGetTypeID

Returns the type identifier for the CFTimeZone opaque type.

```
CFTypeID CFTimeZoneGetTypeID (
   void
);
```

Return Value
The type identifier for the CFTimeZone opaque type.

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

Declared In
`CFTimeZone.h`

CFTimeZoneIsDaylightSavingTime

Returns whether or not a time zone is in daylight savings time at a specified date.

```
Boolean CFTimeZoneIsDaylightSavingTime (
   CFTimeZoneRef tz,
   CFAbsoluteTime at
);
```

Parameters

tz

The time zone to analyze.

at

The date in *tz* to test for daylight savings.

Return Value
`true` if *tz* is in daylight savings time at *at*, otherwise `false`.

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

Declared In
`CFTimeZone.h`

CFTimeZoneResetSystem

Clears the previously determined system time zone, if any.

```
void CFTimeZoneResetSystem (
   void
);
```

Discussion
This function also resets the default time zone if it is the same as the system time zone.

Subsequent calls to `CFTimeZoneCopySystem` (page 627) will attempt to re-determine the system time zone.

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

Declared In
CFTimeZone.h

CFTimeZoneSetAbbreviationDictionary

Sets the abbreviation dictionary to a given dictionary.

```
void CFTimeZoneSetAbbreviationDictionary (
   CFDictionaryRef dict
);
```

Parameters

dict

A dictionary containing key-value pairs for looking up time zone names given their abbreviations. The keys should be CFString objects containing the abbreviations; the values should be CFString objects containing their corresponding geopolitical region names.

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

Declared In
CFTimeZone.h

CFTimeZoneSetDefault

Sets the default time zone for your application the given time zone.

```
void CFTimeZoneSetDefault (
   CFTimeZoneRef tz
);
```

Parameters

tz

The time zone to use as default.

Discussion
There can be only one default time zone, so by setting a new default time zone, you lose the previous one.

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

Declared In
CFTimeZone.h

Data Types

CFTimeZoneNameStyle

Index type for constants used to specify styles of time zone names.

```
typedef CFIndex CFTimeZoneNameStyle;
```

Discussion
For values, see "Time Zone Name Styles" (page 634).

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

Declared In
`CFTimeZone.h`

CFTimeZoneRef

A reference to a CFTimeZone object.

```
typedef const struct __CFTimeZone *CFTimeZoneRef;
```

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

Declared In
`CFDate.h`

Constants

Notification Name

Name of the notification posted when the time zone changes.

```
const CFStringRef kCFTimeZoneSystemTimeZoneDidChangeNotification;
```

Constants
`kCFTimeZoneSystemTimeZoneDidChangeNotification`

Name of the notification posted when the system time zone changes.

The object of the notification is the previous system time zone object. This notification carries no user info.

Keep in mind that there is no order in how notifications are delivered to observers; frameworks or other parts of your code may also be observing this notification to take their own actions, and these may not have occurred by the time you receive the notification.

Available in Mac OS X v10.5 and later.

Declared in `CFTimeZone.h`.

Declared In
`CFTimeZone.h`

Time Zone Name Styles

Constants to specify styles for time zone names.

```
enum {
    kCFTimeZoneNameStyleStandard,
    kCFTimeZoneNameStyleShortStandard,
    kCFTimeZoneNameStyleDaylightSaving,
    kCFTimeZoneNameStyleShortDaylightSaving
};
```

Constants

`kCFTimeZoneNameStyleStandard`
Specifies the standard name style for a time zone.

Available in Mac OS X v10.5 and later.

Declared in `CFTimeZone.h`.

`kCFTimeZoneNameStyleShortStandard`
Specifies the short standard name style for a time zone.

Available in Mac OS X v10.5 and later.

Declared in `CFTimeZone.h`.

`kCFTimeZoneNameStyleDaylightSaving`
Specifies the daylight saving name style for a time zone.

Available in Mac OS X v10.5 and later.

Declared in `CFTimeZone.h`.

`kCFTimeZoneNameStyleShortDaylightSaving`
Specifies the short daylight saving name style for a time zone.

Available in Mac OS X v10.5 and later.

Declared in `CFTimeZone.h`.

Discussion

These constants are used with the function `CFTimeZoneCopyLocalizedName` (page 626).

Declared In

`CFTimeZone.h`

CFTree Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFTree.h
Companion guide	Collections Programming Topics for Core Foundation

Overview

You use CFTree to create tree structures that represent hierarchical organizations of information. In such structures, each tree node has exactly one parent tree (except for the root tree, which has no parent) and can have multiple children. Each CFTree object in the structure has a context associated with it; this context includes some program-defined data as well as callbacks that operate on that data. The program-defined data is often used as the basis for determining where CFTree objects fit within the structure. All CFTree objects are mutable.

You create a CFTree object using the `CFTreeCreate` (page 640) function. This function takes an allocator and pointer to a `CFTreeGetContext` (page 642) structure as parameters. The `CFTreeContext` (page 650) structure contains the program-defined data and callbacks needed to describe, retain, and release that data. If you do not implement these callbacks, your program-defined data will not be retained or released when trees are added and removed from a parent.

Each CFTree object has a parent and list of children, all of which may be `NULL`. CFTree provides functions for adding and removing tree objects from the tree structure. Use the `CFTreeAppendChild` (page 639), `CFTreeInsertSibling` (page 644), or `CFTreePrependChild` (page 645) functions to add trees to a tree structure, and the `CFTreeRemove` (page 645) or `CFTreeRemoveAllChildren` (page 646) functions to remove trees.

For the purposes of memory management, CFTree can be thought of as a collection. Typically the only object that retains a child tree is its parent. Usually, therefore, when you remove a child tree from a tree, the child tree is destroyed. If you want to use a child tree after you remove it from its parent, you should retain the child tree first, prior to removing it.

Releasing a tree releases its child trees, and all of their child trees (recursively). Note also that the final release of a tree (when its retain count decreases to zero) causes all of its child trees, and all of their child trees (recursively), to be destroyed, regardless of their retain counts. Releasing a child that is still in a tree is therefore a programming error, and may cause your application to crash.

You can use any of the get functions (functions containing the word “Get”) to obtain the parent, children, or attributes of a tree. For example, use `CFTreeGetChildAtIndex` (page 641) to obtain a child of a tree at a specified location. In common with other Core Foundation “Get” functions, these functions do not retain the tree that is returned. If you are making other modifications to the tree, you should either retain or make a deep copy of the child tree returned.

You can apply a function to all children of a tree using the `CFTreeApplyFunctionToChildren` (page 640) function, and sort children of a tree using the `CFTreeSortChildren` (page 647) function.

Functions by Task

Creating Trees

`CFTreeCreate` (page 640)
Creates a new CFTree object.

Modifying a Tree

`CFTreeAppendChild` (page 639)
Adds a new child to a tree as the last in its list of children.

`CFTreeInsertSibling` (page 644)
Inserts a new sibling after a given tree.

`CFTreeRemoveAllChildren` (page 646)
Removes all the children of a tree.

`CFTreePrependChild` (page 645)
Adds a new child to the specified tree as the first in its list of children.

`CFTreeRemove` (page 645)
Removes a tree from its parent.

`CFTreeSetContext` (page 646)
Replaces the context of a tree by releasing the old information pointer and retaining the new one.

Sorting a Tree

`CFTreeSortChildren` (page 647)
Sorts the immediate children of a tree using a specified comparator function.

Examining a Tree

`CFTreeFindRoot` (page 641)
Returns the root tree of a given tree.

`CFTreeGetChildAtIndex` (page 641)
Returns the child of a tree at the specified index.

`CFTreeGetChildCount` (page 642)
Returns the number of children in a tree.

`CFTreeGetChildren` (page 642)
Fills a buffer with children from the tree.

`CFTreeGetContext` (page 642)
Returns the context of the specified tree.

`CFTreeGetFirstChild` (page 643)
Returns the first child of a tree.

`CFTreeGetNextSibling` (page 643)
Returns the next sibling, adjacent to a given tree, in the parent's children list.

`CFTreeGetParent` (page 644)
Returns the parent of a given tree.

Performing an Operation on Tree Elements

`CFTreeApplyFunctionToChildren` (page 640)
Calls a function once for each immediate child of a tree.

Getting the Tree Type ID

`CFTreeGetTypeID` (page 644)
Returns the type identifier of the CFTree opaque type.

Functions

CFTreeAppendChild

Adds a new child to a tree as the last in its list of children.

```
void CFTreeAppendChild (
   CFTreeRef tree,
   CFTreeRef newChild
);
```

Parameters

tree
The tree to which to add *newChild*.

newChild
The child tree to add to *tree*. If this parameter is a tree which is already a child of any other tree, the behavior is undefined.

Discussion
When a child tree is added to another tree, the child tree is retained by its new parent.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeApplyFunctionToChildren

Calls a function once for each immediate child of a tree.

```
void CFTreeApplyFunctionToChildren (
   CFTreeRef tree,
   CFTreeApplierFunction applier,
   void *context
);
```

Parameters

tree

The tree to operate upon.

applier

The callback function to call once for each child in *tree*. The function must be able to apply to all the values in the tree.

context

A pointer-sized program-defined value that is passed to the applier function, but is otherwise unused by this function.

Discussion

Note that the applier only operates one level deep—it does not operate on descendants further removed than the immediate children of a tree. If the tree is mutable, it is unsafe for the applied function to change the contents of the tree.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeCreate

Creates a new CFTree object.

```
CFTreeRef CFTreeCreate (
   CFAllocatorRef allocator,
   const CFTreeContext *context
);
```

Parameters

allocator

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

context

The `CFTreeContext` (page 650) structure to be copied and used as the context of the new tree. The information pointer will be retained by the tree if a retain function is provided. If this value is not a valid C pointer to a `CFTreeContext` structure-sized block of storage, the result is undefined. If the version number of the storage is not a valid `CFTreeContext` version number, the result is undefined.

Return Value

A new CFTree object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeFindRoot

Returns the root tree of a given tree.

```
CFTreeRef CFTreeFindRoot (
   CFTreeRef tree
);
```

Parameters

tree
The tree to examine.

Return Value
The root of *tree* where root is defined as a tree without a parent. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeGetChildAtIndex

Returns the child of a tree at the specified index.

```
CFTreeRef CFTreeGetChildAtIndex (
   CFTreeRef tree,
   CFIndex idx
);
```

Parameters

tree
The tree to examine.

idx
The index of the child obtain. The value must be less than the number of children in *tree*.

Return Value
The child tree at *idx*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeGetChildCount

Returns the number of children in a tree.

```
CFIndex CFTreeGetChildCount (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value

The number of children in *tree*.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeGetChildren

Fills a buffer with children from the tree.

```
void CFTreeGetChildren (
   CFTreeRef tree,
   CFTreeRef *children
);
```

Parameters

tree

The tree to examine.

children

The C array of pointer-sized values to be filled with the children from *tree*. This value must be a valid pointer to a C array of at least the size of the number of children in *tree*. Use the `CFTreeGetChildCount` (page 642) function to obtain the number of children in *tree*. You are responsible for retaining and releasing the returned objects as needed.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFTree.h

CFTreeGetContext

Returns the context of the specified tree.

```
void CFTreeGetContext (
   CFTreeRef tree,
   CFTreeContext *context
);
```

Parameters

tree

The tree to examine.

context

The `CFTreeContext` (page 650) structure to be filled in with the context of the specified tree. This value must be a valid C pointer to a `CFTreeContext` structure-sized block of storage. If the version number of the storage is not a valid `CFTreeContext` structure version number, the result is undefined.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeGetFirstChild

Returns the first child of a tree.

```
CFTreeRef CFTreeGetFirstChild (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value

The first child of *tree*. Ownership follows the Get Rule.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeGetNextSibling

Returns the next sibling, adjacent to a given tree, in the parent's children list.

```
CFTreeRef CFTreeGetNextSibling (
   CFTreeRef tree
);
```

Parameters

tree

The tree to examine.

Return Value
The next sibling, adjacent to *tree*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeGetParent

Returns the parent of a given tree.

```
CFTreeRef CFTreeGetParent (
   CFTreeRef tree
);
```

Parameters
tree
The tree to examine.

Return Value
The parent of *tree*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeGetTypeID

Returns the type identifier of the CFTree opaque type.

```
CFTypeID CFTreeGetTypeID (
   void
);
```

Return Value
The type identifier of the CFTree opaque type.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFTree.h

CFTreeInsertSibling

Inserts a new sibling after a given tree.

```
void CFTreeInsertSibling (
   CFTreeRef tree,
   CFTreeRef newSibling
);
```

Parameters

tree

The tree after which to insert *newSibling*. *tree* must have a parent.

newSibling

The sibling to add. *newSibling* must not have a parent.

Discussion

When a child tree is added to another tree, the child tree is retained by its new parent.

If you want to manipulate an existing tree structure, since *newSibling* must not have a parent you need to remove a tree from its parent in order to move it to a new position. If you do this, you should retain the tree before you actually remove it from its parent (see CFTreeRemove (page 645)).

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreePrependChild

Adds a new child to the specified tree as the first in its list of children.

```
void CFTreePrependChild (
   CFTreeRef tree,
   CFTreeRef newChild
);
```

Parameters

tree

The tree to which to add *newChild*.

newChild

The child tree to add to *tree*. This value must not be a child of another tree.

Discussion

When a child tree is added to another tree, the child tree is retained by its new parent.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRemove

Removes a tree from its parent.

```
void CFTreeRemove (
   CFTreeRef tree
);
```

Parameters

tree

The tree to remove from its parent.

Discussion

When a child tree is removed from its parent, the parent releases it. If you want to use the child after you have removed it, you should ensure you retain it before removing it from its parent.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRemoveAllChildren

Removes all the children of a tree.

```
void CFTreeRemoveAllChildren (
   CFTreeRef tree
);
```

Parameters

tree

The tree to modify.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeSetContext

Replaces the context of a tree by releasing the old information pointer and retaining the new one.

```
void CFTreeSetContext (
   CFTreeRef tree,
   const CFTreeContext *context
);
```

Parameters

tree

The tree to modify.

context

The `CFTreeContext` (page 650) structure to be copied and used as the context of the new tree. The information pointer will be retained by the tree if a retain function is provided. If this value is not a valid C pointer to a `CFTreeContext` structure-sized block of storage, the result is undefined. If the version number of the storage is not a valid `CFTreeContext` version number, the result is undefined.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

CFTreeSortChildren

Sorts the immediate children of a tree using a specified comparator function.

```
void CFTreeSortChildren (
   CFTreeRef tree,
   CFComparatorFunction comparator,
   void *context
);
```

Parameters

tree

The tree to sort.

comparator

The function with a comparator function type signature which is used in the sort operation to compare children of the tree. The children of the tree are sorted from least to greatest according to this function.

context

A pointer-sized program-defined value that is passed to the comparator function, but is otherwise unused by this function.

Discussion
Note that the comparator only operates one level deep and does not operate on descendants further removed than the immediate children of a tree node.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFTree.h`

Callbacks

CFTreeApplierFunction

Type of the callback function used by the CFTree apply function.

```
typedef void (*CFTreeApplierFunction) (
   const void *value,
   void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   const void *value,
   void *context
);
```

Parameters

value

The current child of a tree that is being iterated.

context

The program-defined context parameter that was passed to the applier function.

Discussion

This callback is used by the `CFTreeApplyFunctionToChildren` (page 640) applier function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeCopyDescriptionCallBack

Callback function used to provide a description of the program-defined information pointer.

```
typedef CFStringRef (*CFTreeCopyDescriptionCallBack) (
   const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFStringRef MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 650) structure.

Return Value

A textual description of *info*. The caller is responsible for releasing this object.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeReleaseCallBack

Callback function used to release a previously retained program-defined information pointer.

```
typedef void (*CFTreeReleaseCallBack) (
   const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 650) structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRetainCallBack

Callback function used to retain a program-defined information pointer.

```
typedef const void *(*CFTreeRetainCallBack) (
const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
   const void *info
);
```

Parameters

info

The program-supplied information pointer provided in a `CFTreeContext` (page 650) structure.

Return Value

The value to use whenever the information pointer is retained, which is usually the *info* parameter passed to this callback, but may be a different value if a different value should be used.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

Data Types

CFTreeContext

Structure containing program-defined data and callbacks for a CFTree object.

```
struct CFTreeContext {
    CFIndex version;
    void *info;
    CFTreeRetainCallBack retain;
    CFTreeReleaseCallBack release;
    CFTreeCopyDescriptionCallBack copyDescription;
};
typedef struct CFTreeContext CFTreeContext;
```

Fields

`version`
: The version number of the structure type being passed in as a parameter to a CFTree creation function. This structure is version `0`.

`info`
: A C pointer to a program-defined block of data, referred to as the information pointer.

`retain`
: The callback used to retain the `info` field. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. This value may be `NULL`.

`release`
: The callback used to release a previously retained `info` field. If this parameter is not a pointer to a function of the correct prototype, the behavior is undefined. This value may be `NULL`.

`copyDescription`
: The callback used to provide a description of the `info` field.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFTreeRef

A reference to a CFTree object.

```
typedef struct __CFTree *CFTreeRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFTree.h`

CFType Reference

Derived From:	None
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBase.h CFString.h
Companion guides	Core Foundation Design Concepts Memory Management Programming Guide for Core Foundation

Overview

All other Core Foundation opaque types derive from CFType. The functions, callbacks, data types, and constants defined for CFType can be used by any derived opaque type. Hence, CFType functions are referred to as "polymorphic functions." You use CFType functions to retain and release objects, to compare and inspect objects, get descriptions of objects and opaque types, and to get object allocators.

Functions by Task

Memory Management

`CFGetAllocator` (page 654)
Returns the allocator used to allocate a Core Foundation object.

`CFGetRetainCount` (page 654)
Returns the reference count of a Core Foundation object.

`CFMakeCollectable` (page 656)
Makes a newly-allocated Core Foundation object eligible for garbage collection.

`CFRelease` (page 657)
Releases a Core Foundation object.

`CFRetain` (page 657)
Retains a Core Foundation object.

Determining Equality

`CFEqual` (page 653)
Determines whether two Core Foundation objects are considered equal.

Hashing

`CFHash` (page 656)
Returns a code that can be used to identify an object in a hashing structure.

Miscellaneous Functions

`CFCopyDescription` (page 652)
Returns a textual description of a Core Foundation object.

`CFCopyTypeIDDescription` (page 653)
Returns a textual description of a Core Foundation type, as identified by its type ID, which can be used when debugging.

`CFGetTypeID` (page 655)
Returns the unique identifier of an opaque type to which a Core Foundation object belongs.

`CFShow` (page 658)
Prints a description of a Core Foundation object to stderr.

Functions

CFCopyDescription

Returns a textual description of a Core Foundation object.

```
CFStringRef CFCopyDescription (
   CFTypeRef cf
);
```

Parameters

cf
The CFType object (a generic reference of type `CFTypeRef` (page 660)) from which to derive a description.

Return Value
A string that contains a description of *cf*. Ownership follows the Create Rule.

Discussion
The nature of the description differs by object. For example, a description of a CFArray object would include descriptions of each of the elements in the collection.

You can use this function for debugging Core Foundation objects in your code. Note, however, that the description for a given object may be different in different releases of the operating system. Do *not* create dependencies in your code on the content or format of the information returned by this function.

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

Related Sample Code
CFPrefsDumper
FSFileOperation

MoreSCF

Declared In
CFBase.h

CFCopyTypeIDDescription

Returns a textual description of a Core Foundation type, as identified by its type ID, which can be used when debugging.

```
CFStringRef CFCopyTypeIDDescription (
   CFTypeID type_id
);
```

Parameters

theType
An integer of type CFTypeID (page 659) that uniquely identifies a Core Foundation opaque type.

Return Value
A string containing a type description. Ownership follows the Create Rule.

Discussion
You can use this function for debugging Core Foundation objects in your code. Note, however, that the description for a given object may be different in different releases of the operating system. Do *not* create dependencies in your code on the content or format of the information returned by this function.

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

Related Sample Code
MoreSCF

Declared In
CFBase.h

CFEqual

Determines whether two Core Foundation objects are considered equal.

```
Boolean CFEqual (
   CFTypeRef cf1,
   CFTypeRef cf2
);
```

Parameters

cf1
A CFType object to compare to *cf2*.

cf2
A CFType object to compare to *cf1*.

Return Value
`true` if *cf1* and *cf2* are of the same type and considered equal, otherwise `false`.

Discussion
Equality is something specific to each Core Foundation opaque type. For example, two CFNumber objects are equal if the numeric values they represent are equal. Two CFString objects are equal if they represent identical sequences of characters, regardless of encoding.

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

Related Sample Code
audioburntest
bulkerase
databurntest
MoreSCF
Spotlight

Declared In
`CFBase.h`

CFGetAllocator

Returns the allocator used to allocate a Core Foundation object.

```
CFAllocatorRef CFGetAllocator (
   CFTypeRef cf
);
```

Parameters

cf
The CFType object to examine.

Return Value
The allocator used to allocate memory for *cf*.

Discussion
When you are creating a Core Foundation object sometimes you want to ensure that the block of memory allocated for the object is from the same allocator used for another object. One way to do this is to reuse the allocator assigned to an existing Core Foundation object when you call a "creation" function.

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

Declared In
`CFBase.h`

CFGetRetainCount

Returns the reference count of a Core Foundation object.

```
CFIndex CFGetRetainCount (
   CFTypeRef cf
);
```

Parameters

cf

The CFType object to examine.

Return Value
A number representing the reference count of *cf*.

Discussion
You increment the reference count using the `CFRetain` (page 657) function, and decrement the reference count using the `CFRelease` (page 657) function.

This function may useful for debugging memory leaks. You normally do not use this function, otherwise.

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

Related Sample Code
AuthForAll
BSDLLCTest
MoreSCF

Declared In
`CFBase.h`

CFGetTypeID

Returns the unique identifier of an opaque type to which a Core Foundation object belongs.

```
CFTypeID CFGetTypeID (
   CFTypeRef cf
);
```

Parameters

cf

The CFType object to examine.

Return Value
A value of type `CFTypeID` (page 659) that identifies the opaque type of *cf*.

Discussion
This function returns a value that uniquely identifies the opaque type of any Core Foundation object. You can compare this value with the known `CFTypeID` (page 659) identifier obtained with a "GetTypeID" function specific to a type, for example `CFDateGetTypeID` (page 181). These values might change from release to release or platform to platform.

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

Related Sample Code
HID Calibrator
HID Config Save

HID Dumper
HID Utilities
MoreSCF

Declared In
`CFBase.h`

CFHash

Returns a code that can be used to identify an object in a hashing structure.

```
CFHashCode CFHash (
   CFTypeRef cf
);
```

Parameters

cf

A CFType object to examine.

Return Value
An integer of type `CFHashCode` (page 659) that represents a hashing value for *cf*.

Discussion
Two objects that are equal (as determined by the `CFEqual` (page 653) function) have the same hashing value. However, the converse is not true: two objects with the same hashing value might not be equal. That is, hashing values are not necessarily unique.

The hashing value for an object might change from release to release or from platform to platform.

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

Declared In
`CFBase.h`

CFMakeCollectable

Makes a newly-allocated Core Foundation object eligible for garbage collection.

```
CFTypeRef CFMakeCollectable (
    CFTypeRef cf
);
```

Parameters

cf

A CFType object to make collectable. This value must not be `NULL`.

Return Value
cf.

Discussion
For more details, see *Garbage Collection Programming Guide*.

Special Considerations

If *cf* is NULL, this will cause a runtime error and your application will crash.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

AutomatorHandsOn
CIRAWFilterSample

Declared In

CFBase.h

CFRelease

Releases a Core Foundation object.

```
void CFRelease (
    CFTypeRef cf
);
```

Parameters

cf

A CFType object to release. This value must not be NULL.

Discussion

If the retain count of *cf* becomes zero the memory allocated to the object is deallocated and the object is destroyed. If you create, copy, or explicitly retain (see the CFRetain (page 657) function) a Core Foundation object, you are responsible for releasing it when you no longer need it (see *Memory Management Programming Guide for Core Foundation*).

Special Considerations

If *cf* is NULL, this will cause a runtime error and your application will crash.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

HID Calibrator
HID Config Save
HID Dumper
HID Utilities
ImageClient

Declared In

CFBase.h

CFRetain

Retains a Core Foundation object.

```
CFTypeRef CFRetain (
   CFTypeRef cf
);
```

Parameters

cf

The CFType object to retain. This value must not be `NULL`

Return Value
The input value, *cf*.

Discussion
You should retain a Core Foundation object when you receive it from elsewhere (that is, you did not create or copy it) and you want it to persist. If you retain a Core Foundation object you are responsible for releasing it (see *Memory Management Programming Guide for Core Foundation*).

Special Considerations

If *cf* is `NULL`, this will cause a runtime error and your application will crash.

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

Related Sample Code
audioburntest
bulkerase
databurntest
MoreSCF
SampleHardwarePlugIn

Declared In
`CFBase.h`

CFShow

Prints a description of a Core Foundation object to stderr.

```
void CFShow (
   CFTypeRef obj
);
```

Parameters

obj

A Core Foundation object derived from CFType. If `obj` is not a Core Foundation object, an assertion is raised.

Discussion
The output is printed to the standard I/O standard error (stderr).

This function is useful as a debugging aid for Core Foundation objects. Because these objects are based on opaque types, it is difficult to examine their contents directly. However, the opaque types implement `description` function callbacks that return descriptions of their objects. This function invokes these callbacks.

Special Considerations

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

```
(gdb) call (void) CFShow(string)
Hello World
```

You can also incorporate calls to `CFShow` in a test version of your code to print out "snapshots" of Core Foundation objects to the console.

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

Related Sample Code
FSFileOperation
HID Dumper
MoreSCF
ProfileSystem
USBPrivateDataSample

Declared In
`CFString.h`

Data Types

CFHashCode

A type for hash codes returned by the `CFHash` function.

```
typedef unsigned long CFHashCode;
```

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

Declared In
`CFBase.h`

CFTypeID

A type for unique, constant integer values that identify particular Core Foundation opaque types.

```
typedef unsigned long CFTypeID;
```

Discussion
Defines a type identifier in Core Foundation. A type ID is an integer that identifies the opaque type to which a Core Foundation object “belongs.” You use type IDs in various contexts, such as when you are operating on heterogeneous collections. Core Foundation provides programmatic interfaces for obtaining and evaluating type IDs.

Because the value for a type ID can change from release to release, your code should not rely on stored or hard-coded type IDs nor should it hard-code any observed properties of a type ID (such as, for example, it being a small integer).

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

Declared In
`CFBase.h`

CFTypeRef

An untyped "generic" reference to any Core Foundation object.

```
typedef const void * CFTypeRef;
```

Discussion
The `CFTypeRef` type is the base type defined in Core Foundation. It is used as the type and return value in several polymorphic functions. It is a generic object reference that acts as a placeholder for other true Core Foundation objects.

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

Declared In
`CFBase.h`

CFURL Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFURL.h

Overview

`CFURL` provides facilities for creating, parsing, and dereferencing URL strings. `CFURL` is useful to applications that need to use URLs to access resources, including local files.

A `CFURL` object is composed of two parts—a base URL, which can be `NULL`, and a string that is resolved relative to the base URL. A `CFURL` object whose string is fully resolved without a base URL is considered absolute; all others are considered relative.

`CFURL` fails to create an object if the string passed is not well-formed (that is, if it does not comply with RFC 2396). Examples of cases that will not succeed are strings containing space characters and high-bit characters. If a function fails to create a `CFURL` object, it returns *`NULL`*, which you must be prepared to handle. If you create `CFURL` objects using file system paths, you should use the `CFURLCreateFromFileSystemRepresentation` (page 685) and `CFURLCreateFromFileSystemRepresentationRelativeToBase` (page 685) functions, which handle the subtle differences between URL paths and file system paths.

For functions that read and write data from a URL, see *Core Foundation URL Access Utilities Reference*

`CFURL` is "toll-free bridged" with its Cocoa Foundation counterpart, NSURL. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. In other words, in a method where you see an `NSURL *` parameter, you can pass in a `CFURLRef`, and in a function where you see a `CFURLRef` parameter, you can pass in an `NSURL` instance. This also applies to concrete subclasses of `NSURL`. See Integrating Carbon and Cocoa in Your Application for more information on toll-free bridging.

Functions by Task

Creating a CFURL

`CFURLCopyAbsoluteURL` (page 666)
Creates a new `CFURL` object by resolving the relative portion of a URL against its base.

CFURLCreateAbsoluteURLWithBytes (page 677)
Creates a new CFURL object by resolving the relative portion of a URL, specified as bytes, against its given base URL.

CFURLCreateByResolvingBookmarkData (page 679)
Returns a new URL made by resolving bookmark data.

CFURLCreateCopyAppendingPathComponent (page 680)
Creates a copy of a given URL and appends a path component.

CFURLCreateCopyAppendingPathExtension (page 681)
Creates a copy of a given URL and appends a path extension.

CFURLCreateCopyDeletingLastPathComponent (page 682)
Creates a copy of a given URL with the last path component deleted.

CFURLCreateCopyDeletingPathExtension (page 682)
Creates a copy of a given URL with its last path extension removed.

CFURLCreateFilePathURL (page 684)
Initializes and returns a newly created CFURL object as a file URL with a specified path.

CFURLCreateFileReferenceURL (page 684)
Returns a new file reference URL that points to the same resource as a specified URL.

CFURLCreateFromFileSystemRepresentation (page 685)
Creates a new CFURL object for a file system entity using the native representation.

CFURLCreateFromFileSystemRepresentationRelativeToBase (page 685)
Creates a CFURL object from a native character string path relative to a base URL.

CFURLCreateFromFSRef (page 686)
Creates a URL from a given directory or file.

CFURLCreateWithBytes (page 691)
Creates a CFURL object using a given character bytes.

CFURLCreateWithFileSystemPath (page 692)
Creates a CFURL object using a local file system path string.

CFURLCreateWithFileSystemPathRelativeToBase (page 693)
Creates a CFURL object using a local file system path string relative to a base URL.

CFURLCreateWithString (page 693)
Creates a CFURL object using a given CFString object.

Accessing the Parts of a URL

CFURLCanBeDecomposed (page 665)
Determines if the given URL conforms to RFC 1808 and therefore can be decomposed.

CFURLCopyFileSystemPath (page 666)
Returns the path portion of a given URL.

CFURLCopyFragment (page 667)
Returns the fragment from a given URL.

CFURLCopyHostName (page 668)
Returns the host name of a given URL.

CFURLCopyLastPathComponent (page 668)
Returns the last path component of a given URL.

`CFURLCopyNetLocation` (page 669)
Returns the net location portion of a given URL.

`CFURLCopyParameterString` (page 670)
Returns the parameter string from a given URL.

`CFURLCopyPassword` (page 670)
Returns the password of a given URL.

`CFURLCopyPath` (page 671)
Returns the path portion of a given URL.

`CFURLCopyPathExtension` (page 672)
Returns the path extension of a given URL.

`CFURLCopyQueryString` (page 672)
Returns the query string of a given URL.

`CFURLCopyResourceSpecifier` (page 674)
Returns any additional resource specifiers after the path.

`CFURLCopyScheme` (page 675)
Returns the scheme portion of a given URL.

`CFURLCopyStrictPath` (page 675)
Returns the path portion of a given URL.

`CFURLCopyUserName` (page 676)
Returns the user name from a given URL.

`CFURLGetPortNumber` (page 698)
Returns the port number from a given URL.

`CFURLHasDirectoryPath` (page 699)
Determines if a given URL's path represents a directory.

Converting URLs to Other Representations

`CFURLCreateData` (page 683)
Creates a `CFData` object containing the content of a given URL.

`CFURLCreateStringByAddingPercentEscapes` (page 688)
Creates a copy of a string, replacing certain characters with the equivalent percent escape sequence based on the specified encoding.

`CFURLCreateStringByReplacingPercentEscapes` (page 689)
Creates a new string by replacing any percent escape sequences with their character equivalent.

`CFURLCreateStringByReplacingPercentEscapesUsingEncoding` (page 690)
Creates a new string by replacing any percent escape sequences with their character equivalent.

`CFURLGetFileSystemRepresentation` (page 696)
Fills a buffer with the file system's native string representation of a given URL's path.

`CFURLGetFSRef` (page 697)
Converts a given URL to a file or directory object.

`CFURLGetString` (page 698)
Returns the URL as a `CFString` object.

Getting URL Properties

`CFURLGetBaseURL` (page 694)
Returns the base URL of a given URL if it exists.

`CFURLGetBytes` (page 695)
Returns by reference the byte representation of a URL object.

`CFURLGetByteRangeForComponent` (page 695)
Returns the range of the specified component in the bytes of a URL.

`CFURLGetTypeID` (page 699)
Returns the type identifier for the `CFURL` opaque type.

`CFURLResourceIsReachable` (page 700)
Returns whether the resource pointed to by a file URL can be reached.

Getting and Setting File System Resource Properties

`CFURLClearResourcePropertyCache` (page 665)
Clears all cached resource property values of a given URL.

`CFURLClearResourcePropertyCacheForKey` (page 666)
Discards a cached property value for a given key of a given URL.

`CFURLCopyResourcePropertiesForKeys` (page 673)
Returns any number of resource property values of a URL as a dictionary.

`CFURLCopyResourcePropertyForKey` (page 674)
Returns the value of a given resource property of a given URL.

`CFURLCreateResourcePropertiesForKeysFromBookmarkData` (page 687)
Returns the resource values for properties identified by a specified array of keys contained in specified bookmark data.

`CFURLCreateResourcePropertyForKeyFromBookmarkData` (page 687)
Returns the value of a resource property from specified bookmark data.

`CFURLSetResourcePropertiesForKeys` (page 700)
Sets resource properties of a URL specified by a given dictionary of keys and values.

`CFURLSetResourcePropertyForKey` (page 701)
Sets the resource property of the URL specified by a given key to a given value.

`CFURLSetTemporaryResourcePropertyForKey` (page 701)
Sets the resource property of the URL specified by a given key to a given value without writing the assignment to the backing store.

Working with Bookmark Data

`CFURLCreateBookmarkData` (page 677)
Returns bookmark data for a URL, created with specified options and resource values.

`CFURLCreateBookmarkDataFromAliasRecord` (page 678)
Initializes and returns bookmark data derived from an alias record.

`CFURLCreateBookmarkDataFromFile` (page 679)
Initializes and returns bookmark data derived from a file pointed to by a specified URL.

`CFURLWriteBookmarkDataToFile` (page 702)
Creates an alias file on disk at a specified location with specified bookmark data.

Functions

CFURLCanBeDecomposed

Determines if the given URL conforms to RFC 1808 and therefore can be decomposed.

```
Boolean CFURLCanBeDecomposed (
   CFURLRef anURL
);
```

Parameters

anURL
The `CFURL` object to test.

Return Value

`true` if *anURL* conforms to RFC 1808, `false` otherwise.

Discussion

If a `CFURL` object can be decomposed, you can retrieve separately each of the four components (scheme, net location, path, and resource specifier), as well as the base URL.

Relative URLs are permitted to have only paths (or a variety of other configurations); these are considered decomposable if their base URL is decomposable. If no base URL is present, they are considered decomposable.

Availability

Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In

`CFURL.h`

CFURLClearResourcePropertyCache

Clears all cached resource property values of a given URL.

```
void CFURLClearResourcePropertyCache (
    CFURLRef url
);
```

Parameters

url
The URL.

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFURL.h`

CFURLClearResourcePropertyCacheForKey

Discards a cached property value for a given key of a given URL.

```
void CFURLClearResourcePropertyCacheForKey (
    CFURLRef url,
    CFStringRef key
);
```

Parameters

url

The URL.

key

The property value key.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLCopyAbsoluteURL

Creates a new CFURL object by resolving the relative portion of a URL against its base.

```
CFURLRef CFURLCopyAbsoluteURL (
   CFURLRef relativeURL
);
```

Parameters

relativeURL

The CFURL object to resolve.

Return Value
A new CFURL object, or NULL if *relativeURL* cannot be made absolute. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
ImageClient

Declared In
CFURL.h

CFURLCopyFileSystemPath

Returns the path portion of a given URL.

```
CFStringRef CFURLCopyFileSystemPath (
   CFURLRef anURL,
   CFURLPathStyle pathStyle
);
```

Parameters

anURL

The `CFURL` object whose path you want to obtain.

pathStyle

The operating system path style to be used to create the path. See Path Style (page 712) for a list of possible values.

Return Value

The URL's path in the format specified by *pathStyle*. Ownership follows the Create Rule.

Discussion

This function returns the URL's path as a file system path for a given path style.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioBurn

DisplayURL

MoreAppleEvents

RecordAudioToFile

SeeMyFriends

Declared In

`CFURL.h`

CFURLCopyFragment

Returns the fragment from a given URL.

```
CFStringRef CFURLCopyFragment (
   CFURLRef anURL,
   CFStringRef charactersToLeaveEscaped
);
```

Parameters

anURL

The `CFURL` object whose fragment you want to obtain.

charactersToLeaveEscaped

Characters whose percent escape sequences, such as `%20` for a space character, you want to leave intact. Pass `NULL` to specify that no percent escapes be replaced, or the empty string (`CFSTR("")`) to specify that all be replaced.

Return Value

The fragment, or `NULL` if no fragment exists. Ownership follows the Create Rule.

Discussion
A fragment is the text following a "#". These are generally used to indicate locations within a single file. This function removes all percent escape sequences except those for characters specified in *charactersToLeaveEscaped*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLCopyHostName

Returns the host name of a given URL.

```
CFStringRef CFURLCopyHostName (
   CFURLRef anURL
);
```

Parameters

anURL
The CFURL object to examine.

Return Value
The host name of *anURL*. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
ImageClient

Declared In
CFURL.h

CFURLCopyLastPathComponent

Returns the last path component of a given URL.

```
CFStringRef CFURLCopyLastPathComponent (
   CFURLRef url
);
```

Parameters

url
The CFURL object to examine.

Return Value
The last path component of *url*. Ownership follows the Create Rule.

Discussion
Note that if there is no last path component, this function returns an empty string. In the code sample shown in Listing 48-1, `lastPathComponent` is an empty string.

Listing 48-1 Code sample illustrating CFURLCopyLastPathComponent

```
CFStringRef urlString = CFSTR("http://www.apple.com");
CFURLRef url = CFURLCreateWithString(NULL, urlString, NULL);
CFStringRef lastPathComponent = CFURLCopyLastPathComponent (url);
```

If `urlString` were created with `CFSTR("http://www.apple.com/")`, then `lastPathComponent` would be a `CFString` object containing the character "/".

See also `CFURLCopyPathExtension` (page 672).

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFFTPSample
DisplayURL
OpenALExample
RecentItems
SimpleAudioExtraction

Declared In
`CFURL.h`

CFURLCopyNetLocation

Returns the net location portion of a given URL.

```
CFStringRef CFURLCopyNetLocation (
   CFURLRef anURL
);
```

Parameters
anURL
The `CFURL` object to examine.

Return Value
The net location of *anURL*, or `NULL` if the URL cannot be decomposed (doesn't conform to RFC 1808). Ownership follows the Create Rule.

Discussion
The URL net location is the portion of the URL that identifies the network address of the resource. It includes the optional username and password, as well as the target machine's IP address or host name.

This function leaves any percent escape sequences intact.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
`CFURL.h`

CFURLCopyParameterString

Returns the parameter string from a given URL.

```
CFStringRef CFURLCopyParameterString (
   CFURLRef anURL,
   CFStringRef charactersToLeaveEscaped
);
```

Parameters

anURL

The `CFURL` object to examine.

charactersToLeaveEscaped

Characters whose percent escape sequences, such as `%20` for a space character, you want to leave intact. Pass `NULL` to specify that no percent escapes be replaced, or the empty string (`CFSTR("")`) to specify that all be replaced.

Return Value
The parameter string (as defined in RFC 1738), or `NULL` if no parameter string exists. Ownership follows the Create Rule.

Discussion
This function removes all percent escape sequences except those for characters specified in *charactersToLeaveEscaped*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
`CFURL.h`

CFURLCopyPassword

Returns the password of a given URL.

```
CFStringRef CFURLCopyPassword (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value

The password, or `NULL` if no password exists. In some cases, this function may also return the empty string (`CFSTR("")`) if no password exists. You should consider `NULL` and the empty string to be equivalent. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

DisplayURL

Declared In

`CFURL.h`

CFURLCopyPath

Returns the path portion of a given URL.

```
CFStringRef CFURLCopyPath (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value

The path of *anURL*, or `NULL` if the URL cannot be decomposed (doesn't conform to RFC 1808). Ownership follows the Create Rule.

Discussion

This function does not resolve the URL against its base and replaces all percent escape sequences. This function's return value includes any leading slash (giving the path the normal POSIX appearance), if present. If this behavior is not appropriate, use `CFURLCopyStrictPath` (page 675) whose return value omits any leading slash. You may also want to use the function `CFURLCopyFileSystemPath` (page 666), which returns the URL's path as a file system path for the given path style. If the path is to be passed to file system calls, you may also want to use the function `CFURLGetFileSystemRepresentation` (page 696), which returns a C string.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

ConvertFile

DisplayURL

ImageClient
SampleDS

Declared In
CFURL.h

CFURLCopyPathExtension

Returns the path extension of a given URL.

```
CFStringRef CFURLCopyPathExtension (
   CFURLRef url
);
```

Parameters

url

The CFURL object to examine.

Return Value
The path extension of *url*, or NULL if no extension exists. Ownership follows the Create Rule.

Discussion
The path extension is the portion of the last path component which follows the final period, if there is one. For example, for http:/www.apple.com/developer/macosx.today.html, the extension is html, and for http:/www.apple.com/developer, there is no path extension.

See also CFURLCopyLastPathComponent (page 668).

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLCopyQueryString

Returns the query string of a given URL.

```
CFStringRef CFURLCopyQueryString (
   CFURLRef anURL,
   CFStringRef charactersToLeaveEscaped
);
```

Parameters

anURL

The CFURL object to examine.

charactersToLeaveEscaped
Characters whose percent escape sequences, such as `%20` for a space character, you want to leave intact. Pass `NULL` to specify that no percent escapes be replaced, or the empty string (`CFSTR("")`) to specify that all be replaced.

Return Value
The query string, or `NULL` if no parameter string exists. Ownership follows the Create Rule.

Discussion
This function removes all percent escape sequences except those for characters specified in *charactersToLeaveEscaped*.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
`CFURL.h`

CFURLCopyResourcePropertiesForKeys

Returns any number of resource property values of a URL as a dictionary.

```
CFDictionaryRef CFURLCopyResourcePropertiesForKeys (
   CFURLRef url,
   CFArrayRef keys,
   CFErrorRef *error
);
```

Parameters
url
The URL.

keys
The property value keys that values are requested for.

error
The error that occurred in the case that the bookmark data cannot be created.

Return Value
A dictionary of resource property values, or `NULL` if an error occurs.

Discussion
Values are fetched synchronously from the resource backing store unless they are already cached.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCopyResourcePropertyForKey

Returns the value of a given resource property of a given URL.

```
Boolean CFURLCopyResourcePropertyForKey (
   CFURLRef url,
   CFStringRef key,
   void *propertyValueTypeRefPtr,
   CFErrorRef *error
);
```

Parameters

url

The URL.

key

The property value key that the value is requested for.

propertyValueTypeRefPtr

The pointer that is populated with the result.

error

The error that occurred in the case that the bookmark data cannot be created.

Return Value

`true` if *propertyValueTypeRefPtr* is successfully populated; otherwise, `false`.

Discussion

Values are fetched synchronously from the resource backing store unless they are already cached.

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFURL.h`

CFURLCopyResourceSpecifier

Returns any additional resource specifiers after the path.

```
CFStringRef CFURLCopyResourceSpecifier (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value

The resource specifiers. Ownership follows the Create Rule.

Discussion

This function leaves any percent escape sequences intact. For decomposable URLs, this function returns everything after the path. For URLs that cannot be decomposed, this function returns everything except the scheme itself.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
`CFURL.h`

CFURLCopyScheme

Returns the scheme portion of a given URL.

```
CFStringRef CFURLCopyScheme (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value
The scheme of *anURL*. Ownership follows the Create Rule.

Discussion
The URL scheme is the portion of the URL specifying the transport type. For example `http`, `ftp`, and `rtsp` are schemes. This function leaves any percent escape sequences intact.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
ImageClient

Declared In
`CFURL.h`

CFURLCopyStrictPath

Returns the path portion of a given URL.

```
CFStringRef CFURLCopyStrictPath (
   CFURLRef anURL,
   Boolean *isAbsolute
);
```

Parameters

anURL

The `CFURL` object to examine.

isAbsolute

On return, indicates whether the path of *anURL* is absolute.

Return Value
The path of *anURL*, or NULL if the URL cannot be decomposed (doesn't conform to RFC 1808). Ownership follows the Create Rule.

Discussion
This function does not resolve the URL against its base and replaces all percent escape sequences. This function's return value does not include a leading slash and uses *isAbsolute* to report whether the URL's path is absolute. If this behavior is not appropriate, use the CFURLCopyPath (page 671) function whose return value includes the leading slash (giving the path the normal POSIX appearance). You may also want to use the CFURLCopyFileSystemPath (page 666) function, which returns the URL's path as a file system path for the given path style. If the path is to be passed to file system calls, you may also want to use the function CFURLGetFileSystemRepresentation (page 696), which returns a C string.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLCopyUserName

Returns the user name from a given URL.

```
CFStringRef CFURLCopyUserName (
   CFURLRef anURL
);
```

Parameters

anURL
The CFURL object to examine.

Return Value
The user name, or NULL if no user name exists. In some cases, this function may also return the empty string (CFSTR("")) if no username exists. You should consider NULL and the empty string to be equivalent. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLCreateAbsoluteURLWithBytes

Creates a new `CFURL` object by resolving the relative portion of a URL, specified as bytes, against its given base URL.

```
CFURLRef CFURLCreateAbsoluteURLWithBytes (
   CFAllocatorRef alloc,
   const UInt8 *relativeURLBytes,
   CFIndex length,
   CFStringEncoding encoding,
   CFURLRef baseURL,
   Boolean useCompatibilityMode
);
```

Parameters

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

relativeURLBytes
: The character bytes that represent a relative URL to convert into a `CFURL` object.

length
: The number of bytes in *relativeURLBytes*.

encoding
: The string encoding of the `relativeURLBytes` string. This encoding is also used to interpret percent escape sequences.

baseURL
: The URL to which *relativeURLBytes* is relative.

useCompatibilityMode
: If `true`, the rules historically used on the web are used to resolve the string specified by the *relativeURLBytes* parameter against *baseURL*. These rules are generally listed in the RFC as optional or alternate interpretations. Otherwise, the strict rules from the RFC are used.

Return Value

A new `CFURL` object, or `NULL` if *relativeURLBytes* cannot be made absolute. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFURL.h`

CFURLCreateBookmarkData

Returns bookmark data for a URL, created with specified options and resource values.

```
CFDataRef CFURLCreateBookmarkData (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFURLBookmarkCreationOptions options,
   CFArrayRef resourcePropertiesToInclude,
   CFURLRef relativeToURL,
   CFErrorRef *error
);
```

Parameters

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

url
The URL that bookmark data is being created for.

options
Options taken into account when creating the bookmark data.

resourcePropertiesToInclude
An array of names of URL resource properties.

relativeToURL
The URL that the bookmark data is relative to.

error
The error that occurred in the case that the bookmark data cannot be created.

Return Value
The bookmark data for the URL.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCreateBookmarkDataFromAliasRecord

Initializes and returns bookmark data derived from an alias record.

```
CFDataRef CFURLCreateBookmarkDataFromAliasRecord (
   CFAllocatorRef allocatorRef,
   CFDataRef aliasRecordDataRef
);
```

Parameters

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

aliasRecordDataRef
The alias record.

Return Value
The bookmark data for the alias record.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCreateBookmarkDataFromFile

Initializes and returns bookmark data derived from a file pointed to by a specified URL.

```
CFDataRef CFURLCreateBookmarkDataFromFile (
   CFAllocatorRef allocator,
   CFURLRef fileURL,
   CFErrorRef *errorRef
);
```

Parameters

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

fileURL
The file URL.

errorRef
The error that occurred in the case that the bookmark data cannot be created.

Return Value
The bookmark data for the file, or `NULL` if an error occurs.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCreateByResolvingBookmarkData

Returns a new URL made by resolving bookmark data.

```
CFURLRef CFURLCreateByResolvingBookmarkData (
   CFAllocatorRef allocator,
   CFDataRef bookmark,
   CFURLBookmarkResolutionOptions options,
   CFURLRef relativeToURL,
   CFArrayRef resourcePropertiesToInclude,
   Boolean *isStale,
   CFErrorRef *error
);
```

Parameters

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

bookmark
The bookmark data the URL is derived from.

options
Options taken into account when resolving the bookmark data.

relativeToURL
The base URL that the bookmark data is relative to. Can be NULL.

resourcePropertiesToInclude
An array of resource properties to include when creating the URL. Can be NULL.

isStale
If YES, the bookmark data is stale.

error
The error that occurred in the case that the URL cannot be created.

Return Value
A new URL made by resolving *bookmark*, or NULL if an error occurs.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLCreateCopyAppendingPathComponent

Creates a copy of a given URL and appends a path component.

```
CFURLRef CFURLCreateCopyAppendingPathComponent (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFStringRef pathComponent,
   Boolean isDirectory
);
```

Parameters

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

url
The CFURL object to which to append a path component.

pathComponent
The path component to append to *url*.

isDirectory
A Boolean value that specifies whether the string is treated as a directory path when resolving against relative path components. Pass true if the new component indicates a directory, false otherwise.

Return Value
A copy of *url* appended with *pathComponent*. Ownership follows the Create Rule.

Discussion
The *isDirectory* argument specifies whether or not the new path component points to a file or a to directory. Note that the URL syntax for a directory and for a file at otherwise the same location are slightly different—directory URLs must end in "/". If you have the URL `http://www.apple.com/foo/` and you append the path component `bar`, then if *isDirectory* is `YES` then the resulting URL is `http://www.apple.com/foo/bar/`, whereas if *isDirectory* is `NO` then the resulting URL is `http://www.apple.com/foo/bar`. This difference is particularly important if you resolve another URL against this new URL. `file.html` relative to `http://www.apple.com/foo/bar` is `http://www.apple.com/foo/file.html`, whereas `file.html` relative to `http://www.apple.com/foo/bar/` is `http://www.apple.com/foo/bar/file.html`.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
CFFTPSample
simpleJavaLauncher
SpellingChecker CarbonCocoa Bundled
SpellingChecker-CarbonCocoa

Declared In
`CFURL.h`

CFURLCreateCopyAppendingPathExtension

Creates a copy of a given URL and appends a path extension.

```
CFURLRef CFURLCreateCopyAppendingPathExtension (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFStringRef extension
);
```

Parameters

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

url
The `CFURL` object to which to append a path extension.

extension
The extension to append to *url*.

Return Value
A copy of *url* appended with *extension*. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
`CFURL.h`

CFURLCreateCopyDeletingLastPathComponent

Creates a copy of a given URL with the last path component deleted.

```
CFURLRef CFURLCreateCopyDeletingLastPathComponent (
   CFAllocatorRef allocator,
   CFURLRef url
);
```

Parameters

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

url
The `CFURL` object whose last path component you want to delete.

Return Value
A copy of *url* with the last path component deleted. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
HID Utilities Source
ImageClient

Declared In
`CFURL.h`

CFURLCreateCopyDeletingPathExtension

Creates a copy of a given URL with its last path extension removed.

```
CFURLRef CFURLCreateCopyDeletingPathExtension (
   CFAllocatorRef allocator,
   CFURLRef url
);
```

Parameters

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

url
The `CFURL` object whose path extension you want to delete.

Return Value
A copy of *url* with its last path extension removed. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
OpenALExample

Declared In
CFURL.h

CFURLCreateData

Creates a CFData object containing the content of a given URL.

```
CFDataRef CFURLCreateData (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFStringEncoding encoding,
   Boolean escapeWhitespace
);
```

Parameters

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

url
The URL to convert into a CFData object.

encoding
The string encoding to use when converting *url* into a CFData object.

escapeWhitespace
true if you want to escape whitespace characters in the URL, false otherwise.

Return Value
A new CFData object containing the content of *url*. Ownership follows the Create Rule.

Discussion
This function escapes any character that is not 7-bit ASCII with the byte-code for the given encoding. If *escapeWhitespace* is true, whitespace characters (' ', '\t', '\r', '\n') will be escaped as well. This is desirable if you want to embed the URL into a larger text stream like HTML.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFURL.h

CFURLCreateFilePathURL

Initializes and returns a newly created CFURL object as a file URL with a specified path.

```
CFURLRef CFURLCreateFilePathURL (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFErrorRef *error
);
```

Parameters

allocator

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

url

The path.

error

The error that occurred in the case that the URL cannot be created.

Return Value

A CFURL object initialized with *url*, or `NULL` if an error occurs.

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFURL.h`

CFURLCreateFileReferenceURL

Returns a new file reference URL that points to the same resource as a specified URL.

```
CFURLRef CFURLCreateFileReferenceURL (
   CFAllocatorRef allocator,
   CFURLRef url,
   CFErrorRef *error
);
```

Parameters

allocator

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

url

The URL.

error

The error that occurred in the case that the URL cannot be created.

Return Value

The new file reference URL, or `NULL` if an error occurs.

Availability

Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLCreateFromFileSystemRepresentation

Creates a new CFURL object for a file system entity using the native representation.

```
CFURLRef CFURLCreateFromFileSystemRepresentation (
   CFAllocatorRef allocator,
   const UInt8 *buffer,
   CFIndex bufLen,
   Boolean isDirectory
);
```

Parameters

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

buffer
The character bytes to convert into a CFURL object. This should be the path as you would use in POSIX function calls.

bufLen
The number of bytes in the buffer.

isDirectory
A Boolean value that specifies whether the string is treated as a directory path when resolving against relative path components—true if the pathname indicates a directory, false otherwise.

Return Value
A new CFURL object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioQueueTools
CFFTPSample
ConvertFile
MemoryBasedBundle
OpenCL NBody Simulation Example

Declared In
CFURL.h

CFURLCreateFromFileSystemRepresentationRelativeToBase

Creates a CFURL object from a native character string path relative to a base URL.

```
CFURLRef CFURLCreateFromFileSystemRepresentationRelativeToBase (
   CFAllocatorRef allocator,
   const UInt8 *buffer,
   CFIndex bufLen,
   Boolean isDirectory,
   CFURLRef baseURL
);
```

Parameters

allocator

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

buffer

The character bytes to convert into a `CFURL` object. This should be the path as you would use in POSIX function calls.

bufLen

The number of bytes in the buffer.

isDirectory

A Boolean value that specifies whether the string is treated as a directory path when resolving against relative path components. Pass `true` if the pathname indicates a directory, `false` otherwise.

baseURL

The URL against which to resolve the path.

Return Value

A new `CFURL` object. Ownership follows the Create Rule.

Discussion

This function takes a path name in the form of a native character string, resolves it against a base URL, and returns a new `CFURL` object containing the result.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFURL.h`

CFURLCreateFromFSRef

Creates a URL from a given directory or file.

```
CFURLRef CFURLCreateFromFSRef (
   CFAllocatorRef allocator,
   const struct FSRef *fsRef
);
```

Parameters

allocator

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

fsRef

The file or directory representing the URL.

Return Value
A new `CFURL` object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreSCF
SeeMyFriends
SimpleAudioExtraction
SimpleScriptingPlugin

Declared In
`CFURL.h`

CFURLCreateResourcePropertiesForKeysFromBookmarkData

Returns the resource values for properties identified by a specified array of keys contained in specified bookmark data.

```
CFDictionaryRef CFURLCreateResourcePropertiesForKeysFromBookmarkData (
   CFAllocatorRef allocator,
   CFArrayRef resourcePropertiesToReturn,
   CFDataRef bookmark
);
```

Parameters

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

`resourcePropertiesToReturn`
An array of names of URL resource properties.

`bookmark`
The bookmark data the resource values are derived from.

Return Value
A dictionary of the requested resource values contained in *`bookmarkData`*.

Discussion
This function does not attempt to resolve the bookmark data or perform I/O.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCreateResourcePropertyForKeyFromBookmarkData

Returns the value of a resource property from specified bookmark data.

```
CFTypeRef CFURLCreateResourcePropertyForKeyFromBookmarkData (
   CFAllocatorRef allocator,
   CFStringRef resourcePropertyKey,
   CFDataRef bookmark
);
```

Parameters

allocator

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

resourcePropertyKey

The resource property key.

bookmark

The bookmark data the resource value is derived from.

Return Value
The resource property value.

Discussion
This function does not attempt to resolve the bookmark data or perform I/O.

Availability
Available in Mac OS X v10.6 and later.

Declared In
`CFURL.h`

CFURLCreateStringByAddingPercentEscapes

Creates a copy of a string, replacing certain characters with the equivalent percent escape sequence based on the specified encoding.

```
CFStringRef CFURLCreateStringByAddingPercentEscapes (
   CFAllocatorRef allocator,
   CFStringRef originalString,
   CFStringRef charactersToLeaveUnescaped,
   CFStringRef legalURLCharactersToBeEscaped,
   CFStringEncoding encoding
);
```

Parameters

allocator

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

originalString

The `CFString` object to copy.

charactersToLeaveUnescaped

Characters whose percent escape sequences you want to leave intact. Pass `NULL` to specify that all escape sequences be replaced.

legalURLCharactersToBeEscaped

Legal characters to be escaped. Pass `NULL` to specify that no legal characters be replaced.

encoding

The encoding to use for the translation. If you are uncertain of the correct encoding, you should use UTF-8, which is the encoding designated by RFC 2396 as the correct encoding for use in URLs.

Return Value
A copy of *originalString* replacing certain characters. If it does not need to be modified (no percent escape sequences are missing), this function may merely return *originalString* with its reference count incremented. Ownership follows the Create Rule.

Discussion
The characters escaped are all characters that are not legal URL characters (based on RFC 2396), plus any characters in *legalURLCharactersToBeEscaped*, less any characters in *charactersToLeaveUnescaped*. To simply correct any non-URL characters in an otherwise correct URL string, pass `NULL` for the *allocator*, *charactersToLeaveEscaped*, and *legalURLCharactersToBeEscaped* parameters, and `kCFStringEncodingUTF8` (page 594) as the *encoding* parameter.

It may be difficult to use this function to "clean up" unescaped or partially escaped URL strings where sequences are unpredictable and you cannot specify *charactersToLeaveUnescaped*. Instead, you can "pre-process" a URL string using CFURLCreateStringByReplacingPercentEscapesUsingEncoding (page 690) then add the escape characters using CFURLCreateStringByAddingPercentEscapes (page 688), as shown in the following code fragment.

```
CFStringRef originalURLString =
CFSTR("http://online.store.com/storefront/?request=get-document&doi=10.1175%2F1520-0426(2005)014%3C1157:DODADS%3E2.0.CO%3B2");
CFStringRef preprocessedString =
    CFURLCreateStringByReplacingPercentEscapesUsingEncoding(kCFAllocatorDefault,
 originalURLString, CFSTR(""), kCFStringEncodingUTF8);
CFStringRef urlString =
    CFURLCreateStringByAddingPercentEscapes(kCFAllocatorDefault,
preprocessedString, NULL, NULL, kCFStringEncodingUTF8);
url = CFURLCreateWithString(kCFAllocatorDefault, urlString, NULL);
```

Availability
Available in CarbonLib v1.3 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFNetworkHTTPDownload

Declared In
`CFURL.h`

CFURLCreateStringByReplacingPercentEscapes

Creates a new string by replacing any percent escape sequences with their character equivalent.

```
CFStringRef CFURLCreateStringByReplacingPercentEscapes (
   CFAllocatorRef allocator,
   CFStringRef originalString,
   CFStringRef charactersToLeaveEscaped
);
```

Parameters

allocator

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

originalString

The `CFString` object to be copied and modified.

charactersToLeaveEscaped

Characters whose percent escape sequences, such as `%20` for a space character, you want to leave intact. Pass `NULL` to specify that no percent escapes be replaced, or the empty string (`CFSTR("")`) to specify that all be replaced.

Return Value

A new `CFString` object, or `NULL` if the percent escapes cannot be converted to characters, assuming UTF-8 encoding. If no characters need to be replaced, this function returns the original string with its reference count incremented. Ownership follows the Create Rule.

Availability

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code

CFNetworkHTTPDownload

Declared In

`CFURL.h`

CFURLCreateStringByReplacingPercentEscapesUsingEncoding

Creates a new string by replacing any percent escape sequences with their character equivalent.

```
CFStringRef CFURLCreateStringByReplacingPercentEscapesUsingEncoding (
   CFAllocatorRef allocator,
   CFStringRef origString,
   CFStringRef charsToLeaveEscaped,
   CFStringEncoding encoding
);
```

Parameters

allocator

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

originalString

The `CFString` object to be copied and modified.

charactersToLeaveEscaped

Characters whose percent escape sequences, such as `%20` for a space character, you want to leave intact. Pass `NULL` to specify that no percent escapes be replaced, or the empty string (`CFSTR("")`) to specify that all be replaced.

encoding

Specifies the encoding to use when interpreting percent escapes.

Return Value

A new `CFString` object, or `NULL` if the percent escapes cannot be converted to characters, assuming the encoding given by *encoding*. If no characters need to be replaced, this function returns the original string with its reference count incremented. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFURL.h`

CFURLCreateWithBytes

Creates a `CFURL` object using a given character bytes.

```
CFURLRef CFURLCreateWithBytes (
   CFAllocatorRef allocator,
   const UInt8 *URLBytes,
   CFIndex length,
   CFStringEncoding encoding,
   CFURLRef baseURL
);
```

Parameters

allocator

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

URLBytes

The character bytes to convert into a `CFURL` object.

length

The number of bytes in *URLBytes*.

encoding

The string encoding of the `URLBytes` string. This encoding is also used to interpret percent escape sequences.

baseURL

The URL to which *URLBytes* is relative. Pass `NULL` if *URLBytes* contains an absolute URL or if you want to create a relative URL. If *URLBytes* contains an absolute URL, this parameter is ignored.

Return Value

A new `CFURL` object. Ownership follows the Create Rule.

Discussion

The specified string encoding will be used both to interpret *URLBytes*, and to interpret any percent-escapes within the string.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
RecentItems

Declared In
`CFURL.h`

CFURLCreateWithFileSystemPath

Creates a `CFURL` object using a local file system path string.

```
CFURLRef CFURLCreateWithFileSystemPath (
   CFAllocatorRef allocator,
   CFStringRef filePath,
   CFURLPathStyle pathStyle,
   Boolean isDirectory
);
```

Parameters

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

`filePath`
The path string to convert to a `CFURL` object.

`pathStyle`
The operating system path style used in `filePath`. See Path Style (page 712) for a list of possible values.

`isDirectory`
A Boolean value that specifies whether `filePath` is treated as a directory path when resolving against relative path components. Pass `true` if the pathname indicates a directory, `false` otherwise.

Return Value
A new `CFURL` object. Ownership follows the Create Rule.

Discussion
If *`filePath`* is not absolute, the resulting URL will be considered relative to the current working directory (evaluated when this function is being invoked).

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
From A View to A Movie
From A View to A Picture
ImageBrowserViewAppearance
Quartz EB

Declared In
CFURL.h

CFURLCreateWithFileSystemPathRelativeToBase

Creates a CFURL object using a local file system path string relative to a base URL.

```
CFURLRef CFURLCreateWithFileSystemPathRelativeToBase (
   CFAllocatorRef allocator,
   CFStringRef filePath,
   CFURLPathStyle pathStyle,
   Boolean isDirectory,
   CFURLRef baseURL
);
```

Parameters

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

filePath
The path string to convert to a CFURL object.

pathStyle
The operating system path style used in the *filePath* string. See Path Style (page 712) for a list of possible values.

isDirectory
A Boolean value that specifies whether *filePath* is treated as a directory path when resolving against relative path components. Pass true if the pathname indicates a directory, false otherwise.

baseURL
The base URL against which to resolve the *filePath*.

Return Value
A new CFURL object. Ownership follows the Create Rule.

Discussion
This function takes a path name in the form of a CFString object, resolves it against a base URL, and returns a new CFURL object containing the result.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
Aperture Image Resizer
DisplayURL

Declared In
CFURL.h

CFURLCreateWithString

Creates a CFURL object using a given CFString object.

```
CFURLRef CFURLCreateWithString (
   CFAllocatorRef allocator,
   CFStringRef URLString,
   CFURLRef baseURL
);
```

Parameters

allocator

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

URLString

The `CFString` object containing the URL string.

baseURL

The URL to which `URLString` is relative. Pass `NULL` if *`URLString`* contains an absolute URL or if you want to create a relative URL. If `URLString` contains an absolute URL, `baseURL` is ignored.

Return Value
A new `CFURL` object. Ownership follows the Create Rule.

Discussion
Any escape sequences in *`URLString`* will be interpreted using UTF-8.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AuthForAll
CFFTPSample
DisplayURL
DockBrowser
OpenALExample

Declared In
`CFURL.h`

CFURLGetBaseURL

Returns the base URL of a given URL if it exists.

```
CFURLRef CFURLGetBaseURL (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value
A `CFURL` object representing the base URL of *`anURL`*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLGetByteRangeForComponent

Returns the range of the specified component in the bytes of a URL.

```
CFRange CFURLGetByteRangeForComponent (
   CFURLRef url,
   CFURLComponentType component,
   CFRange *rangeIncludingSeparators
);
```

Parameters

anURL

The URL containing *component*.

component

The type of component in *anURL* whose range you want to obtain. See Component Type (page 710) for possible values.

rangeIncludingSeparators

Specifies the range of *component* including the sequences that separate component from the previous and next components. If there is no previous or next components, this function will match the range of the component itself. If *anURL* does not contain *component*, *rangeIncludingSeparators* is set to the location where the component would be inserted.

Return Value
The range of bytes for *component* in the buffer returned by the CFURLGetBytes (page 695) function. If *anURL* does not contain *component*, the first part of the returned range is set to kCFNotFound (page 803).

Discussion
This function is intended to be used in conjunction with the CFURLGetBytes (page 695) function, since the range returned is only applicable to the bytes returned by CFURLGetBytes (page 695).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
DisplayURL

Declared In
CFURL.h

CFURLGetBytes

Returns by reference the byte representation of a URL object.

```
CFIndex CFURLGetBytes (
   CFURLRef url,
   UInt8 *buffer,
   CFIndex bufferLength
);
```

Parameters

anURL

The URL object to convert to a byte representation.

buffer

The buffer where you want the bytes to be placed. If the buffer is of insufficient size, returns `-1` and no bytes are placed in buffer. If `NULL` the needed length is computed and returned. The returned bytes are the original bytes from which the URL was created. If the URL was created from a string, the bytes are the bytes of the string encoded via UTF-8.

bufferLength

The number of bytes in *buffer*.

Return Value

Returns the number of bytes in *buffer* that were filled. If the buffer is of insufficient size, returns `-1`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

DisplayURL

Declared In

`CFURL.h`

CFURLGetFileSystemRepresentation

Fills a buffer with the file system's native string representation of a given URL's path.

```
Boolean CFURLGetFileSystemRepresentation (
   CFURLRef url,
   Boolean resolveAgainstBase,
   UInt8 *buffer,
   CFIndex maxBufLen
);
```

Parameters

url

The `CFURL` object whose native file system representation you want to obtain.

resolveAgainstBase

Pass `true` to return an absolute path name.

buffer

A pointer to a character buffer. On return, the buffer holds the native file system's representation of *url*. The buffer is null-terminated. This parameter must be at least *maxBufLen* in size for the file system in question to avoid failures for insufficiently large buffers.

maxBufLen

The maximum number of characters that can be written to *buffer*.

Return Value
`true` if successful, `false` if an error occurred.

Discussion
No more than *`maxBufLen`* bytes are written to *`buffer`*. If *`url`* requires more than *`maxBufLen`* bytes to represent itself, including the terminating null byte, this function returns `false`. To avoid this possible failure, you should pass a buffer with size of at least the maximum path length for the file system in question.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BetterAuthorizationSample
BSDLLCTest
CFPrefTopScores
DisplayURL
EmbededAppleScripts

Declared In
`CFURL.h`

CFURLGetFSRef

Converts a given URL to a file or directory object.

```
Boolean CFURLGetFSRef (
   CFURLRef url,
   struct FSRef *fsRef
);
```

Parameters

`url`
The `CFURL` object to convert to a file or directory object.

`fsRef`
Upon return, contains the file or directory object representing *`url`*.

Return Value
`true` if the conversion was successful, otherwise `false`.

Special Considerations

The function cannot create an `FSRef` object if the path specified by `url` contains an alias. The function can, however, traverse symbolic links.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
BasicInputMethod
CoreTextTest
FileNotification

OutputBins2PDE
QTGraphicsImport

Declared In
`CFURL.h`

CFURLGetPortNumber

Returns the port number from a given URL.

```
SInt32 CFURLGetPortNumber (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to examine.

Return Value
The port number of *anURL*, or `-1` if no port number exists.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
DisplayURL
ImageClient

Declared In
`CFURL.h`

CFURLGetString

Returns the URL as a `CFString` object.

```
CFStringRef CFURLGetString (
   CFURLRef anURL
);
```

Parameters

anURL

The `CFURL` object to convert into a `CFString` object.

Return Value
A string representation of *anURL*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AlbumToSlideshow

DisplayURL
FSMegaInfo
LoginItemsAE

Declared In
`CFURL.h`

CFURLGetTypeID

Returns the type identifier for the `CFURL` opaque type.

```
CFTypeID CFURLGetTypeID (
   void
);
```

Return Value
The type identifier for the `CFURL` opaque type.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
LoginItemsAE

Declared In
`CFURL.h`

CFURLHasDirectoryPath

Determines if a given URL's path represents a directory.

```
Boolean CFURLHasDirectoryPath (
   CFURLRef anURL
);
```

Parameters
anURL
The `CFURL` object to examine.

Return Value
`true` if *anURL* represents a directory, `false` otherwise.

Availability
Available in CarbonLib v1.0 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CFFTPSample
DisplayURL
ImageClient
MoreAppleEvents

Declared In
CFURL.h

CFURLResourceIsReachable

Returns whether the resource pointed to by a file URL can be reached.

```
Boolean CFURLResourceIsReachable (
   CFURLRef url,
   CFErrorRef *error
);
```

Parameters

url
The URL to check.

error
The error that occurred in the case that the resource cannot be reached.

Return Value
`true` if the resource is reachable; otherwise, `false`.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLSetResourcePropertiesForKeys

Sets resource properties of a URL specified by a given dictionary of keys and values.

```
Boolean CFURLSetResourcePropertiesForKeys (
   CFURLRef url,
   CFDictionaryRef keyedPropertyValues,
   CFErrorRef *error
);
```

Parameters

url
The URL.

keyedPropertyValues
A dictionary of resource values to be set.

error
The error that occurred in the case that one or more resource values cannot be set.

Return Value
`true` if all resource values in *keyedPropertyValues* are successfully set; otherwise, `false`.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLSetResourcePropertyForKey

Sets the resource property of the URL specified by a given key to a given value.

```
Boolean CFURLSetResourcePropertyForKey (
   CFURLRef url,
   CFStringRef key,
   CFTypeRef propertValue,
   CFErrorRef *error
);
```

Parameters

url

The URL.

key

The name of one of the URL's resource properties.

propertValue

The value for the resource property defined by *key*.

error

The error that occurred in the case that the resource value cannot be set.

Return Value

`true` if the resource property named *key* is successfully set to *value*; otherwise, `false`.

Discussion

This function writes the new property value out to the backing store.

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFURL.h`

CFURLSetTemporaryResourcePropertyForKey

Sets the resource property of the URL specified by a given key to a given value without writing the assignment to the backing store.

```
void CFURLSetTemporaryResourcePropertyForKey (
   CFURLRef url,
   CFStringRef key,
   CFTypeRef propertyValue
);
```

Parameters

url

The URL.

key

The name of one of the URL's resource properties.

propertyValue

The value for the resource property defined by *key*.

Availability

Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLWriteBookmarkDataToFile

Creates an alias file on disk at a specified location with specified bookmark data.

```
Boolean CFURLWriteBookmarkDataToFile (
   CFDataRef bookmarkRef,
   CFURLRef fileURL,
   CFURLBookmarkFileCreationOptions options,
   CFErrorRef *errorRef
);
```

Parameters

bookmarkRef
The bookmark data containing information for the alias file.

fileURL
The desired location of the alias file.

options
Options taken into account when creating the alias file.

errorRef
The error that occurred in the case that the alias file cannot be created.

Return Value
`true` if the alias file is successfully created; otherwise, `false`.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

Data Types

Bookmark Data Types

CFURLBookmarkCreationOptions

Type for bookmark data creation options.

```
typedef CFOptionFlags CFURLBookmarkCreationOptions;
```

Discussion
See "Bookmark Data Creation Options" (page 704) for possible values.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLBookmarkFileCreationOptions

Type for bookmark file creation options.

```
typedef CFOptionFlags CFURLBookmarkFileCreationOptions;
```

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

CFURLBookmarkResolutionOptions

Type for bookmark data resolution options.

```
typedef CFOptionFlags CFURLBookmarkResolutionOptions;
```

Discussion
See "Bookmark Data Resolution Options" (page 704) for possible values.

Availability
Available in Mac OS X v10.6 and later.

Declared In
CFURL.h

Miscellaneous

CFURLRef

A reference to a CFURL object.

```
typedef const struct __CFURL *CFURLRef;
```

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

Declared In
CFURL.h

Constants

Bookmark Data Constants

Bookmark Data Creation Options

Options used when creating bookmark data.

```
enum {
    kCFURLBookmarkCreationPreferFileIDResolutionMask = ( 1UL << 8 ),
    kCFURLBookmarkCreationMinimalBookmarkMask = ( 1UL << 9 ),
    kCFURLBookmarkCreationSuitableForBookmarkFile = ( 1UL << 10 )
};
```

Constants

`kCFURLBookmarkCreationPreferFileIDResolutionMask`
Option for specifying that an alias created with the bookmark data prefers resolving with its embedded file ID.

`kCFURLBookmarkCreationMinimalBookmarkMask`
Option for specifying that an alias created with the bookmark data be created with minimal information, which may make it smaller but still able to resolve in certain ways.

`kCFURLBookmarkCreationSuitableForBookmarkFile`
Option for specifying that the bookmark data include properties required to create Finder alias files.

Bookmark Data Resolution Options

Options used when resolving bookmark data.

```
enum {
    kCFBookmarkResolutionWithoutUIMask = ( 1UL << 8 ),
    kCFBookmarkResolutionWithoutMountingMask = ( 1UL << 9 ),
};
```

Constants

`kCFBookmarkResolutionWithoutUIMask`
Option for specifying that no UI feedback accompany resolution of the bookmark data.

`kCFBookmarkResolutionWithoutMountingMask`
Option for specifying that no volume should be mounted during resolution of the bookmark data.

File System Constants

Common File System Resource Keys

Keys that are applicable to file system URLs.

```
kCFURLNameKey
kCFURLLocalizedNameKey
kCFURLIsRegularFileKey
kCFURLIsDirectoryKey
kCFURLIsSymbolicLinkKey
kCFURLIsVolumeKey
kCFURLIsPackageKey
kCFURLIsSystemImmutableKey
kCFURLIsUserImmutableKey
kCFURLIsHiddenKey
kCFURLHasHiddenExtensionKey
kCFURLCreationDateKey
kCFURLContentAccessDateKey
kCFURLContentModificationDateKey
kCFURLAttributeModificationDateKey
kCFURLLinkCountKey
kCFURLParentDirectoryURLKey
kCFURLVolumeURLKey
kCFURLTypeIdentifierKey
kCFURLLocalizedTypeDescriptionKey
kCFURLLabelNumberKey
kCFURLLabelColorKey
kCFURLLocalizedLabelKey
kCFURLEffectiveIconKey
kCFURLCustomIconKey
```

Constants

`kCFURLNameKey`

Key for the resource's name in the file system, returned as a `CFString` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLocalizedNameKey`

Key for the resource's localized or extension-hidden name, retuned as a `CFString` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsRegularFileKey`

Key for determining whether the resource is a regular file, as opposed to a directory or a symbolic link. Returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsDirectoryKey`

Key for determining whether the resource is a directory, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsSymbolicLinkKey`

Key for determining whether the resource is a symbolic link, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsVolumeKey`
Key for determining whether the resource is the root directory of a volume, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsPackageKey`
Key for determining whether the resource is a packaged directory, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsSystemImmutableKey`
Key for determining whether the resource's system immutable bit is set, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsUserImmutableKey`
Key for determining whether the resource's user immutable bit is set, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsHiddenKey`
Key for determining whether the resource is normally not displayed to users, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLHasHiddenExtensionKey`
Key for determining whether the resource's extension is normally removed from its localized name, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLCreationDateKey`
Key for the resource's creation date, returned as a `CFDate` object if the volume supports creation dates, or `nil` if creation dates are unsupported.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLContentAccessDateKey`
Key for the last time the resource was accessed, returned as a `CFDate` object if the volume supports access dates, or `nil` if access dates are unsupported.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLContentModificationDateKey`
Key for the last time the resource was modified, returned as a `CFDate` object if the volume supports modification dates, or `nil` if modification dates are unsupported.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLAttributeModificationDateKey`
Key for the last time the resource's attributes were modified, returned as a `CFDate` object if the volume supports attribute modification dates, or `nil` if attribute modification dates are unsupported.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLinkCountKey`
Key for the number of hard links to the resource, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLParentDirectoryURLKey`
Key for the parent directory of the resource, returned as a `CFURL` object, or `nil` if the resource is the root directory of its volume.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeURLKey`
Key for the root directory of the resource's volume, returned as a `CFURL` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLTypeIdentifierKey`
Key for the resource's uniform type identifier (UTI), returned as a `CFString` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLocalizedTypeDescriptionKey`
Key for the resource's localized type description, returned as a `CFString` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLabelNumberKey`
Key for the resource's label number, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLabelColorKey`
Key for the resource's label color, returned as a `CFColorRef` object, or `nil` if the resource has no label color.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLLocalizedLabelKey`
Key for the resource's localized label text, returned as a `CFString` object, or `nil` if the resource has no localized label text.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLEffectiveIconKey`
Key for the resource's normal icon, returned as a `CGImageRef` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLCustomIconKey`
Key for the icon stored with the resource, returned as a `CGImageRef` object, or `nil` if the resource has no custom icon.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

File Property Keys

Keys that apply to properties of files.

```
kCFURLFileSizeKey
kCFURLFileAllocatedSizeKey
kCFURLIsAliasFileKey
```

Constants

`kCFURLFileSizeKey`
Key for the file's size in bytes, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLFileAllocatedSizeKey`
Key for the total size allocated on disk for the file, returned as an `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLIsAliasFileKey`
Key for determining whether the file is an alias, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

Volume Property Keys

Keys that apply to volumes.

```
kCFURLVolumeLocalizedFormatDescriptionKey
kCFURLVolumeTotalCapacityKey
kCFURLVolumeAvailableCapacityKey
kCFURLVolumeResourceCountKey
kCFURLVolumeSupportsPersistentIDsKey
kCFURLVolumeSupportsSymbolicLinksKey
kCFURLVolumeSupportsHardLinksKey
kCFURLVolumeSupportsJournalingKey
kCFURLVolumeIsJournalingKey
kCFURLVolumeSupportsSparseFilesKey
kCFURLVolumeSupportsZeroRunsKey
kCFURLVolumeSupportsCaseSensitiveNamesKey
kCFURLVolumeSupportsCasePreservedNamesKey
```

Constants

`kCFURLVolumeLocalizedFormatDescriptionKey`
Key for the volume's descriptive format name, returned as a `CFString` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeTotalCapacityKey`
Key for the volume's capacity in bytes, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeAvailableCapacityKey`
Key for the volume's available capacity in bytes, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeResourceCountKey`
Key for the total number of resources on the volume, returned as a `CFNumber` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsPersistentIDsKey`
Key for determining whether the volume supports persistent IDs, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsSymbolicLinksKey`
Key for determining whether the volume supports symbolic links, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsHardLinksKey`
Key for determining whether the volume supports hard links, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsJournalingKey`
Key for determining whether the volume supports journaling, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeIsJournalingKey`
Key for determining whether the volume is currently journaling, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsSparseFilesKey`
Key for determining whether the volume supports sparse files, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsZeroRunsKey`
Key for determining whether the volume supports zero runs, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsCaseSensitiveNamesKey`
Key for determining whether the volume supports case-sensitive names, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

`kCFURLVolumeSupportsCasePreservedNamesKey`
Key for determining whether the volume supports case-preserved names, returned as a `CFBoolean` object.

Available in Mac OS X v10.6 and later.

Declared in `CFURL.h`.

Miscellaneous

Component Type

The types of components in a URL.

```
typedef enum {
    kCFURLComponentScheme = 1,
    kCFURLComponentNetLocation = 2,
    kCFURLComponentPath = 3,
    kCFURLComponentResourceSpecifier = 4,
    kCFURLComponentUser = 5,
    kCFURLComponentPassword = 6,
    kCFURLComponentUserInfo = 7,
    kCFURLComponentHost = 8,
    kCFURLComponentPort = 9,
    kCFURLComponentParameterString = 10,
    kCFURLComponentQuery = 11,
    kCFURLComponentFragment = 12
} CFURLComponentType;
typedef enum CFURLPathStyle CFURLPathStyle;
```

Constants

`kCFURLComponentScheme`
The URL's scheme.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentNetLocation`
The URL's network location.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentPath`
The URL's path component.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentResourceSpecifier`
The URL's resource specifier.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentUser`
The URL's user.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentPassword`
The user's password.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentUserInfo`
The user's information.
Available in Mac OS X v10.3 and later.
Declared in `CFURL.h`.

`kCFURLComponentHost`
The URL's host.

Available in Mac OS X v10.3 and later.

Declared in `CFURL.h`.

`kCFURLComponentPort`
The URL's port.

Available in Mac OS X v10.3 and later.

Declared in `CFURL.h`.

`kCFURLComponentParameterString`
The URL's parameter string.

Available in Mac OS X v10.3 and later.

Declared in `CFURL.h`.

`kCFURLComponentQuery`
The URL's query.

Available in Mac OS X v10.3 and later.

Declared in `CFURL.h`.

`kCFURLComponentFragment`
The URL's fragment.

Available in Mac OS X v10.3 and later.

Declared in `CFURL.h`.

Discussion
These constants are used by the `CFURLGetByteRangeForComponent` (page 695) function.

Availability
Available in Mac OS X v10.3 and later.

Path Style

Options you can use to determine how CFURL functions parse a file system path name.

```
enum CFURLPathStyle {
    kCFURLPOSIXPathStyle = 0,
    kCFURLHFSPathStyle = 1,
    kCFURLWindowsPathStyle = 2
};
typedef enum CFURLPathStyle CFURLPathStyle;
```

Constants
`kCFURLPOSIXPathStyle`
Indicates a POSIX style path name. Components are slash delimited. A leading slash indicates an absolute path; a trailing slash is not significant.

Available in Mac OS X v10.0 and later.

Declared in `CFURL.h`.

`kCFURLHFSPathStyle`

Indicates a HFS style path name. Components are colon delimited. A leading colon indicates a relative path, otherwise the first path component denotes the volume.

Available in Mac OS X v10.0 and later.

Declared in `CFURL.h`.

`kCFURLWindowsPathStyle`

Indicates a Windows style path name.

Available in Mac OS X v10.0 and later.

Declared in `CFURL.h`.

CFUserNotification Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFUserNotification.h

Overview

A `CFUserNotification` object presents a simple dialog on the screen and optionally receives feedback from the user. The contents of the dialog can include a header, a message, an icon, text fields, a pop-up button, radio buttons or checkboxes, and up to three ordinary buttons. Use `CFUserNotification` in processes that do not otherwise have user interfaces, but may need occasional interaction with the user.

You create a user notification with the `CFUserNotificationCreate` (page 716) function. You pass in a dictionary whose keys describe the items to place into the dialog. (See "Dialog Description Keys" (page 727) for the list of keys.) A set of flags passed to the function determines, among other things, whether secure text fields are used (such as for password fields), whether radio buttons or checkboxes are used, and which of these buttons are checked by default. You can also specify a timeout for the dialog, in which case the dialog cancels itself if the user does not respond in the allotted time period.

A user notification displays its dialog as soon as it is created. If any reply is required, it may be awaited in one of two ways: either synchronously, using `CFUserNotificationReceiveResponse` (page 722), or asynchronously, using a run loop source created with `CFUserNotificationCreateRunLoopSource` (page 717). `CFUserNotificationReceiveResponse` (page 722) has a timeout parameter that determines how long it will block (zero meaning indefinitely) and it may be called as many times as necessary until a response arrives. If a user notification has not yet received a response, it may be updated with new information or it may be cancelled. User notifications may not be reused.

`CFUserNotification` provides two convenience functions, `CFUserNotificationDisplayNotice` (page 719) and `CFUserNotificationDisplayAlert` (page 718), to display very basic dialogs that either require no response from the user or require only a single button to be pressed, respectively.

Functions

CFUserNotificationCancel

Cancels a user notification dialog.

```
SInt32 CFUserNotificationCancel (
   CFUserNotificationRef userNotification
);
```

Parameters

userNotification

The user notification to cancel.

Return Value

`0` if the cancel was successful; a non-`0` value otherwise.

Discussion

You must cancel a user notification if you want to remove its dialog from the screen before the user dismisses it. It is not sufficient to just release the object.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFUserNotification.h`

CFUserNotificationCheckBoxChecked

Returns a flag used to set or test a checkbox's state.

```
CFOptionFlags CFUserNotificationCheckBoxChecked (
   CFIndex i
);
```

Parameters

idx

The index of the checkbox to set or test. The index corresponds to the order in which the checkbox titles are listed in the `kCFUserNotificationCheckBoxTitlesKey` (page 729) array of the user notification's description dictionary. *idx* must be in the range `0` to `7`.

Return Value

A flag that can be used either to set the state of a checkbox when creating a user notification with `CFUserNotificationCreate` (page 716) or to test a checkbox's state returned in a user notification's response flags, such as from `CFUserNotificationReceiveResponse` (page 722), when the notification dialog is dismissed.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFUserNotification.h`

CFUserNotificationCreate

Creates a CFUserNotification object and displays its notification dialog on screen.

```
CFUserNotificationRef CFUserNotificationCreate (
   CFAllocatorRef allocator,
   CFTimeInterval timeout,
   CFOptionFlags flags,
   SInt32 *error,
   CFDictionaryRef dictionary
);
```

Parameters

allocator

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

timeout

The time to wait before the notification dialog dismisses itself if the user does not respond. If `0`, the notification never times out.

flags

A set of flags describing the type of notification to display. These flags specify an alert level for the notification (see "Alert Levels" (page 725)), determine whether radio buttons or checkboxes are to be used (see "Button Flags" (page 727)), specify which, if any, of these buttons are checked by default (see `CFUserNotificationCheckBoxChecked` (page 716)), specify whether any of the text fields are to be secure text fields (see `CFUserNotificationSecureTextField` (page 723)), and determine which element of a pop-up menu, if present, should be selected by default (see `CFUserNotificationPopUpSelection` (page 722)). Combine these flags together by performing a bitwise-OR operation with all the individual flags.

error

On return contains an integer error code. If `0`, the user notification was successfully created and displayed.

dictionary

A description of the elements to display in the notification dialog. The possible keys are listed in "Dialog Description Keys" (page 727). The dictionary must contain a value for the key `kCFUserNotificationAlertHeaderKey` (page 728), but the other keys are optional.

Return Value

The new CFUserNotification object. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFUserNotification.h`

CFUserNotificationCreateRunLoopSource

Creates a run loop source for a user notification.

```
CFRunLoopSourceRef CFUserNotificationCreateRunLoopSource (
   CFAllocatorRef allocator,
   CFUserNotificationRef userNotification,
   CFUserNotificationCallBack callout,
   CFIndex order
);
```

Parameters

allocator

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

userNotification

The user notification to use.

callout

The callback function to invoke when the user notification dialog is dismissed.

order

A priority index indicating the order in which run loop sources are processed. User notifications currently ignore this parameter. Pass `0` for this value.

Return Value
The new CFRunLoopSource object. Ownership follows the Create Rule.

Discussion
A run loop source needs to be added to a run loop before it can fire and call its callback function. To add the source to a run loop, use `CFRunLoopAddSource` (page 457).

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

Declared In
`CFUserNotification.h`

CFUserNotificationDisplayAlert

Displays a user notification dialog and waits for a user response.

```
SInt32 CFUserNotificationDisplayAlert (
   CFTimeInterval timeout,
   CFOptionFlags flags,
   CFURLRef iconURL,
   CFURLRef soundURL,
   CFURLRef localizationURL,
   CFStringRef alertHeader,
   CFStringRef alertMessage,
   CFStringRef defaultButtonTitle,
   CFStringRef alternateButtonTitle,
   CFStringRef otherButtonTitle,
   CFOptionFlags *responseFlags
);
```

Parameters

timeout

The amount of time to wait for the user to dismiss the notification dialog before the dialog dismisses itself. Pass `0` to have the dialog never time out.

flags
: A set of flags describing the type of notification dialog to display. The value is normally just the alert level from "Alert Levels" (page 725). If you don't want a default button displayed, perform a bitwise-OR operation with the alert level and the constant `kCFUserNotificationNoDefaultButtonFlag` (page 727).

iconURL
: A file URL pointing to the icon to display in the dialog. If `NULL`, a default icon is used based on the notification's alert level specified in *flags*.

soundURL
: Not used.

localizationURL
: A file URL pointing to a bundle that contains localized versions of the strings displayed in the dialog. Can be `NULL`.

alertHeader
: The title of the notification dialog. Cannot be `NULL`.

alertMessage
: The message string to display in the dialog. Can be `NULL`.

defaultButtonTitle
: The title of the default button. If `NULL`, the string `OK` is used.

alternateButtonTitle
: The title of an optional alternate button. Can be `NULL`.

otherButtonTitle
: The title of an optional third button. Can be `NULL`.

responseFlags
: On return, contains flags identifying how the notification was dismissed. See "Response Codes" (page 726) for details.

Return Value
`0` if the cancel was successful; a non-`0` value otherwise.

Discussion
This function blocks the current thread's execution until the dialog is dismissed, either by the user or by timing out.

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

Declared In
`CFUserNotification.h`

CFUserNotificationDisplayNotice

Displays a user notification dialog that does not need a user response.

```
SInt32 CFUserNotificationDisplayNotice (
   CFTimeInterval timeout,
   CFOptionFlags flags,
   CFURLRef iconURL,
   CFURLRef soundURL,
   CFURLRef localizationURL,
   CFStringRef alertHeader,
   CFStringRef alertMessage,
   CFStringRef defaultButtonTitle
);
```

Parameters

timeout

The amount of time to wait for the user to dismiss the notification dialog before the dialog dismisses itself. Pass `0` to have the dialog never time out.

flags

A set of flags describing the type of notification dialog to display. The value is normally just the alert level from "Alert Levels" (page 725). If you don't want a default button displayed, perform a bitwise-OR operation with the alert level and the constant `kCFUserNotificationNoDefaultButtonFlag` (page 727).

iconURL

A file URL pointing to the icon to display in the dialog. If `NULL`, a default icon is used based on the notification's alert level specified in *flags*.

soundURL

Not used.

localizationURL

A file URL pointing to a bundle that contains localized versions of the strings displayed in the dialog. Can be `NULL`.

alertHeader

The title of the notification dialog. Cannot be `NULL`.

alertMessage

The message string to display in the dialog. Can be `NULL`.

defaultButtonTitle

The title of the default button. If `NULL`, the string `OK` is used.

Return Value

`0` if the cancel was successful; a non-`0` value otherwise.

Discussion

This function returns immediately. It does not wait for a user response after displaying the dialog.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

X11CallCarbonAndCocoa

Declared In

`CFUserNotification.h`

CFUserNotificationGetResponseDictionary

Returns the dictionary containing all the text field values from a dismissed notification dialog.

```
CFDictionaryRef CFUserNotificationGetResponseDictionary (
   CFUserNotificationRef userNotification
);
```

Parameters

userNotification
The user notification to use.

Return Value

A dictionary holding the values of all the text fields in *userNotification* when it was dismissed. The values are in an array stored with the key kCFUserNotificationTextFieldValuesKey (page 729). Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFUserNotification.h

CFUserNotificationGetResponseValue

Extracts the values of the text fields from a dismissed notification dialog.

```
CFStringRef CFUserNotificationGetResponseValue (
   CFUserNotificationRef userNotification,
   CFStringRef key,
   CFIndex idx
);
```

Parameters

userNotification
The user notification to use.

key
The dictionary key identifying the text fields to use. Currently, only kCFUserNotificationTextFieldValuesKey (page 729) is supported.

idx
The index of the text field value to return. The index corresponds to the order in which text fields are listed in the kCFUserNotificationTextFieldTitlesKey (page 729) array in the user notification's description dictionary.

Return Value

The value of the text field identified by *key* and *idx*. Ownership follows the Get Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFUserNotification.h

CFUserNotificationGetTypeID

Returns the type identifier for the `CFUserNotification` opaque type.

```
CFTypeID CFUserNotificationGetTypeID (
   void
);
```

Return Value
The type identifier for the `CFUserNotification` opaque type.

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

Declared In
`CFUserNotification.h`

CFUserNotificationPopUpSelection

Returns a flag used to set the selected element of a pop-up menu.

```
CFOptionFlags CFUserNotificationPopUpSelection (
   CFIndex n
);
```

Parameters

idx

The index of the pop-up menu element to select. The index corresponds to the order in which the pop-up menu elements are listed in the `kCFUserNotificationPopUpTitlesKey` (page 729) array of the user notification's description dictionary. *idx* must be in the range `0` to `255`.

Return Value
A flag that can be used to set the selected element of a pop-up menu when creating a user notification with `CFUserNotificationCreate` (page 716).

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

Declared In
`CFUserNotification.h`

CFUserNotificationReceiveResponse

Waits for the user to respond to a notification or for the notification to time out.

```
SInt32 CFUserNotificationReceiveResponse (
   CFUserNotificationRef userNotification,
   CFTimeInterval timeout,
   CFOptionFlags *responseFlags
);
```

Parameters

userNotification

The user notification to use.

timeout

The amount of time to wait for the user to respond to *userNotification* or for the notification to time out. If neither happens before *timeout* passes, this function returns a non-0 value. If *timeout* is 0, the function blocks until the user notification is dismissed.

responseFlags

On return, contains flags identifying how the notification was dismissed, the state of any checkboxes, and the selected element of the pop-up menu. Bits 0-1 of the value hold an identifier for the button pressed by the user (see "Response Codes" (page 726)). Extract the identifier by performing a bitwise-AND operation with 0x3. Bits 8-15 of *responseFlags* hold the state of up to 8 checkboxes or radio buttons, if present. Extract the flags by performing bitwise-AND operations with the return value of `CFUserNotificationCheckBoxChecked` (page 716). Bits 24-31 hold the index number of the element selected in a pop-up menu, if present. Extract the index by performing a 24-bit right shift: `responseFlags >> 24`.

Return Value
0 if the cancel was successful; a non-0 value otherwise.

Discussion
Use this function to poll a user notification for a user response. You can call it any number of times on the same user notification.

To avoid polling and blocking your thread's execution, you can create a run loop source for the user notification with `CFUserNotificationCreateRunLoopSource` (page 717). You will then receive a callback when the dialog is dismissed.

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

Declared In
CFUserNotification.h

CFUserNotificationSecureTextField

Returns a flag used to set the secure state of a text field.

```
CFOptionFlags CFUserNotificationSecureTextField (
   CFIndex i
);
```

Parameters

idx

The index of the text field to make secure. The index corresponds to the order in which the text fields are listed in the `kCFUserNotificationTextFieldTitlesKey` (page 729) array of the user notification's description dictionary. *idx* must be in the range 0 to 7.

Return Value
A flag that can be used to set the secure state of a text field when creating a user notification with `CFUserNotificationCreate` (page 716).

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

Declared In
CFUserNotification.h

CFUserNotificationUpdate

Updates a displayed user notification dialog with new user interface information.

```
SInt32 CFUserNotificationUpdate (
   CFUserNotificationRef userNotification,
   CFTimeInterval timeout,
   CFOptionFlags flags,
   CFDictionaryRef dictionary
);
```

Parameters

userNotification
The user notification to update.

timeout
The new timeout value for the dialog.

flags
A set of flags describing the type of notification to display. See `CFUserNotificationCreate` (page 716) for details.

dictionary
A description of the elements to display in the notification dialog. The possible keys are listed in "Dialog Description Keys" (page 727).

Return Value

`0` if the cancel was successful; a non-`0` value otherwise.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFUserNotification.h`

Callbacks

CFUserNotificationCallBack

Callback invoked when an asynchronous user notification dialog is dismissed.

```
typedef void (*CFUserNotificationCallBack) (
    CFUserNotificationRef userNotification,
    CFOptionFlags responseFlags
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
   CFUserNotificationRef userNotification,
   CFOptionFlags responseFlags
);
```

Parameters

userNotification

The user notification that was dismissed.

responseFlags

On return, contains flags identifying how the notification was dismissed, the state of any checkboxes, and the selected item of the pop-up menu. See `CFUserNotificationReceiveResponse` (page 722) for details.

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

Declared In
`CFUserNotification.h`

Data Types

CFUserNotificationRef

A reference to a user notification object.

```
typedef struct __CFUserNotification *CFUserNotificationRef;
```

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

Declared In
`CFUserNotification.h`

Constants

Alert Levels

Flags identifying the seriousness of a user notification.

```
enum {
    kCFUserNotificationStopAlertLevel = 0,
    kCFUserNotificationNoteAlertLevel = 1,
    kCFUserNotificationCautionAlertLevel = 2,
    kCFUserNotificationPlainAlertLevel= 3
};
```

Constants

`kCFUserNotificationStopAlertLevel`

The notification is very serious.

A stop icon is displayed by default.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationNoteAlertLevel`
The notification is not very serious.
A note icon is displayed by default.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

`kCFUserNotificationCautionAlertLevel`
The notification is somewhat serious.
A caution icon is displayed by default.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

`kCFUserNotificationPlainAlertLevel`
The notification is not serious.
An information icon is displayed by default.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

Discussion
If you specify an icon to display in the dialog, this icon overrides the default icon used for each alert level.

Response Codes

Response codes identifying the button that was pressed to dismiss a notification dialog.

```
enum {
    kCFUserNotificationDefaultResponse = 0,
    kCFUserNotificationAlternateResponse = 1,
    kCFUserNotificationOtherResponse = 2,
    kCFUserNotificationCancelResponse = 3
};
```

Constants
`kCFUserNotificationDefaultResponse`
The default button was pressed.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

`kCFUserNotificationAlternateResponse`
The alternate button was pressed.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

`kCFUserNotificationOtherResponse`
The third button was pressed.
Available in Mac OS X v10.0 and later.
Declared in `CFUserNotification.h`.

`kCFUserNotificationCancelResponse`
No button was pressed and the notification timed out.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

Discussion
To extract this value from the response flags of a dismissed notification (such as returned by `CFUserNotificationReceiveResponse` (page 722)), you must perform a bitwise-AND operation between the returned response flags and `0x3` before comparing the value to these constants.

Button Flags

Flags that alter the display of buttons in a user notification dialog.

```
enum {
    kCFUserNotificationNoDefaultButtonFlag = (1 << 5),
    kCFUserNotificationUseRadioButtonsFlag = (1 << 6)
};
```

Constants
`kCFUserNotificationNoDefaultButtonFlag`
Displays the dialog without the default, alternate, or other buttons.

The dialog remains on screen until it times out or you cancel it with `CFUserNotificationCancel` (page 715). If you provide a title for the default button in the user notification's description dictionary, this flag is ignored and buttons show up normally.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationUseRadioButtonsFlag`
Creates a group of radio buttons instead of checkboxes for the elements in the `kCFUserNotificationCheckBoxTitlesKey` (page 729) array in the user notification's description dictionary.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

Discussion
You specify these flags when you create the user notification with `CFUserNotificationCreate` (page 716).

Dialog Description Keys

Keys used in a user notification's description dictionary, which describes the contents of the notification dialog to display.

```
const CFStringRef kCFUserNotificationIconURLKey;
const CFStringRef kCFUserNotificationSoundURLKey;
const CFStringRef kCFUserNotificationLocalizationURLKey;
const CFStringRef kCFUserNotificationAlertHeaderKey;
const CFStringRef kCFUserNotificationAlertMessageKey;
const CFStringRef kCFUserNotificationDefaultButtonTitleKey;
const CFStringRef kCFUserNotificationAlternateButtonTitleKey;
const CFStringRef kCFUserNotificationOtherButtonTitleKey;
const CFStringRef kCFUserNotificationProgressIndicatorValueKey;
const CFStringRef kCFUserNotificationPopUpTitlesKey;
const CFStringRef kCFUserNotificationTextFieldTitlesKey;
const CFStringRef kCFUserNotificationCheckBoxTitlesKey;
const CFStringRef kCFUserNotificationTextFieldValuesKey;
const CFStringRef kCFUserNotificationPopUpSelectionKey
```

Constants

`kCFUserNotificationIconURLKey`

A file URL pointing to the icon to display in the dialog.

If absent, a default icon based on the alert level is used.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationSoundURLKey`

A file URL pointing to a sound that will be played when the alert appears.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationLocalizationURLKey`

A file URL pointing to a bundle that contains localized versions of the strings displayed in the dialog.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationAlertHeaderKey`

The title of the notification dialog.

This key is required.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationAlertMessageKey`

The message string to display in the dialog.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationDefaultButtonTitleKey`

The title of the default button.

If absent and the dialog is not being created with the `kCFUserNotificationNoDefaultButtonFlag` (page 727) flag, a default button title of `OK` is used.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationAlternateButtonTitleKey`
The title of an optional alternate button.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationOtherButtonTitleKey`
The title of an optional third button.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationProgressIndicatorValueKey`
A value to indicate the progress of an operation.

The value is a number between `0` and `1`, for a "definite" progress indicator, or a Boolean for an "indefinite" progress indicator.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationPopUpTitlesKey`
The list of strings to display in a pop-up menu.

The array cannot have more than 256 elements.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationTextFieldTitlesKey`
The list of titles for all the text fields to display.

If only one text field is to be displayed, you can pass its title string directly without putting it into an array first.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationCheckBoxTitlesKey`
The list of titles for all the checkboxes or radio buttons to display.

The array cannot have more than 8 elements. If only one checkbox is to be displayed, you can pass its title string directly without putting it into an array first.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationTextFieldValuesKey`
The list of values to put into the text fields. If only one text field is to be displayed, you can pass its value string directly without putting it into an array first.

Available in Mac OS X v10.0 and later.

Declared in `CFUserNotification.h`.

`kCFUserNotificationPopUpSelectionKey`
The item that was selected from a pop-up menu.

Available in Mac OS X v10.3 and later.

Declared in `CFUserNotification.h`.

Discussion
When creating the user notification with `CFUserNotificationCreate` (page 716), the description dictionary must have a value for `kCFUserNotificationAlertHeaderKey` (page 728). All other keys are optional.

The button title keys must be given in right-to-left order—you therefore must use the `kCFUserNotificationDefaultButtonTitleKey` constant for the rightmost button even if it is conceptually not a "default" button (for example, if you want a single "Cancel" button that should not have color, should not pulse, and should not have return for a key equivalent). If, however, you set the `kCFUserNotificationNoDefaultButtonFlag`, the rightmost button does not behave as a default button (although it will still be the "default" button in the sense of using `kCFUserNotificationDefaultButtonTitleKey` and `kCFUserNotificationDefaultResponse`). The following code fragment shows how you can create a notification that contains a single "Cancel" button that does not behave as a default button.

```
const void* keys[] = {kCFUserNotificationAlertHeaderKey,
                      kCFUserNotificationProgressIndicatorValueKey,
                      kCFUserNotificationDefaultButtonTitleKey};
const void* values[] = {CFSTR("Progress"),
                        kCFBooleanTrue,
                        CFSTR("Cancel")};
CFDictionaryRef parameters = CFDictionaryCreate(0, keys, values,
        sizeof(keys)/sizeof(*keys), &kCFTypeDictionaryKeyCallBacks,
        &kCFTypeDictionaryValueCallBacks);
SInt32 err = 0;
CFUserNotificationCreate(kCFAllocatorDefault, 0,
        kCFUserNotificationPlainAlertLevel |
kCFUserNotificationNoDefaultButtonFlag,
        &err, parameters);
```

If you set the `kCFUserNotificationNoDefaultButtonFlag` flag and do not specify a value for `kCFUserNotificationDefaultButtonTitleKey`, the notification will have no buttons.

CFUUID Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFUUID.h
Companion guide	Plug-ins

Overview

CFUUID objects are used by plug-ins to uniquely identify types, interfaces, and factories. When creating a new type, host developers must generate UUIDs to identify the type as well as its interfaces and factories.

UUIDs (Universally Unique Identifiers), also known as GUIDs (Globally Unique Identifiers) or IIDs (Interface Identifiers), are 128-bit values guaranteed to be unique. A UUID is made unique over both space and time by combining a value unique to the computer on which it was generated—usually the Ethernet hardware address—and a value representing the number of 100-nanosecond intervals since October 15, 1582 at 00:00:00.

The standard format for UUIDs represented in ASCII is a string punctuated by hyphens, for example `68753A44-4D6F-1226-9C60-0050E4C00067`. The hex representation looks, as you might expect, like a list of numerical values preceded by `0x`. For example, `0xD7, 0x36, 0x95, 0x0A, 0x4D, 0x6E, 0x12, 0x26, 0x80, 0x3A, 0x00, 0x50, 0xE4, 0xC0, 0x00, 0x67`. To use a UUID, you simply create it and then copy the resulting strings into your header and C language source files. Because a UUID is expressed simply as an array of bytes, there are no endianness considerations for different platforms.

You can create a CFUUID object, and thereby generate a UUID, using any one of the `CFUUIDCreate...` functions. Use the `CFUUIDGetConstantUUIDWithBytes` (page 736) function if you want to declare a UUID constant in a `#define` statement. You can get the raw bytes of an existing CFUUID object using the `CFUUIDGetUUIDBytes` (page 738) function.

Functions by Task

Creating CFUUID Objects

`CFUUIDCreate` (page 732)
Creates a Universally Unique Identifier (UUID) object.

`CFUUIDCreateFromString` (page 733)
Creates a CFUUID object for a specified string.

`CFUUIDCreateFromUUIDBytes` (page 733)
Creates a CFUUID object from raw UUID bytes.

`CFUUIDCreateWithBytes` (page 735)
Creates a CFUUID object from raw UUID bytes.

Getting Information About CFUUID Objects

`CFUUIDCreateString` (page 734)
Returns the string representation of a specified CFUUID object.

`CFUUIDGetConstantUUIDWithBytes` (page 736)
Returns a CFUUID object from raw UUID bytes.

`CFUUIDGetUUIDBytes` (page 738)
Returns the value of a UUID object as raw bytes.

Getting the CFUUID Type Identifier

`CFUUIDGetTypeID` (page 738)
Returns the type identifier for all CFUUID objects.

Functions

CFUUIDCreate

Creates a Universally Unique Identifier (UUID) object.

```
CFUUIDRef CFUUIDCreate (
   CFAllocatorRef alloc
);
```

Parameters

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

Return Value
A new CFUUID object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AutomatorHandsOn
People
SimpleScriptingPlugin
SimpleStickies

Declared In
CFUUID.h

CFUUIDCreateFromString

Creates a CFUUID object for a specified string.

```
CFUUIDRef CFUUIDCreateFromString (
   CFAllocatorRef alloc,
   CFStringRef uuidStr
);
```

Parameters

alloc

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

uuidStr

A string containing a UUID. The standard format for UUIDs represented in ASCII is a string punctuated by hyphens, for example `68753A44-4D6F-1226-9C60-0050E4C00067`.

Return Value
A new CFUUID object, or if a CFUUID object of the same value already exists, the existing instance with its reference count incremented. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
iSpendPlugin
QuickLookSketch
Spotlight
SpotlightFortunes

Declared In
CFUUID.h

CFUUIDCreateFromUUIDBytes

Creates a CFUUID object from raw UUID bytes.

```
CFUUIDRef CFUUIDCreateFromUUIDBytes (
   CFAllocatorRef alloc,
   CFUUIDBytes bytes
);
```

Parameters

alloc

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

bytes

Raw UUID bytes to use to create the CFUUID object.

Return Value
A new CFUUID object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
iSpendPlugin
QuickLookSketch
Spotlight
SpotlightFortunes

Declared In
`CFUUID.h`

CFUUIDCreateString

Returns the string representation of a specified CFUUID object.

```
CFStringRef CFUUIDCreateString (
   CFAllocatorRef alloc,
   CFUUIDRef uuid
);
```

Parameters

alloc

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

uuid

The CFUUID object whose string representation to obtain.

Return Value
The string representation of *uuid*. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
AutomatorHandsOn
IdentitySample
SimpleScriptingPlugin
SimpleStickies
Watcher

Declared In
`CFUUID.h`

CFUUIDCreateWithBytes

Creates a CFUUID object from raw UUID bytes.

```
CFUUIDRef CFUUIDCreateWithBytes (
   CFAllocatorRef alloc,
   UInt8 byte0,
   UInt8 byte1,
   UInt8 byte2,
   UInt8 byte3,
   UInt8 byte4,
   UInt8 byte5,
   UInt8 byte6,
   UInt8 byte7,
   UInt8 byte8,
   UInt8 byte9,
   UInt8 byte10,
   UInt8 byte11,
   UInt8 byte12,
   UInt8 byte13,
   UInt8 byte14,
   UInt8 byte15
);
```

Parameters

alloc

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

byte0

Raw byte number `0`.

byte1

Raw byte number `1`.

byte2

Raw byte number `2`.

byte3

Raw byte number `3`.

byte4

Raw byte number `4`.

byte5

Raw byte number `5`.

byte6

Raw byte number `6`.

byte7

Raw byte number `7`.

byte8

Raw byte number `8`.

byte9

Raw byte number `9`.

byte10

Raw byte number `10`.

byte11
Raw byte number 11.

byte12
Raw byte number 12.

byte13
Raw byte number 13.

byte14
Raw byte number 14.

byte15
Raw byte number 15.

Return Value
A new CFUUID object, or, if a CFUUID object of the same value already exists, the existing instance with its reference count incremented. Ownership follows the Create Rule.

Discussion
.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
CFUUID.h

CFUUIDGetConstantUUIDWithBytes

Returns a CFUUID object from raw UUID bytes.

```
CFUUIDRef CFUUIDGetConstantUUIDWithBytes (
   CFAllocatorRef alloc,
   UInt8 byte0,
   UInt8 byte1,
   UInt8 byte2,
   UInt8 byte3,
   UInt8 byte4,
   UInt8 byte5,
   UInt8 byte6,
   UInt8 byte7,
   UInt8 byte8,
   UInt8 byte9,
   UInt8 byte10,
   UInt8 byte11,
   UInt8 byte12,
   UInt8 byte13,
   UInt8 byte14,
   UInt8 byte15
);
```

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

byte0
Raw byte number 0.

byte1
Raw byte number 1.

byte2
Raw byte number 2.

byte3
Raw byte number 3.

byte4
Raw byte number 4.

byte5
Raw byte number 5.

byte6
Raw byte number 6.

byte7
Raw byte number 7.

byte8
Raw byte number 8.

byte9
Raw byte number 9.

byte10
Raw byte number 10.

byte11
Raw byte number 11.

byte12
Raw byte number 12.

byte13
Raw byte number 13.

byte14
Raw byte number 14.

byte15
Raw byte number 15.

Return Value
A CFUUID object. Ownership follows the Get Rule.

Discussion
This function can be used in headers to declare a UUID constant with `#define`.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFUUID.h`

CFUUIDGetTypeID

Returns the type identifier for all CFUUID objects.

```
CFTypeID CFUUIDGetTypeID (
   void
);
```

Return Value
The type identifier for the CFUUID opaque type.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFUUID.h`

CFUUIDGetUUIDBytes

Returns the value of a UUID object as raw bytes.

```
CFUUIDBytes CFUUIDGetUUIDBytes (
   CFUUIDRef uuid
);
```

Parameters

uuid
The CFUUID object to examine.

Return Value
The value of *uuid* represented as raw bytes.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Related Sample Code
Deva_Example
HID Manager Basics
SampleUSBMIDIDriver
SCSIOldAndNew
USBPrivateDataSample

Declared In
`CFUUID.h`

Data Types

CFUUIDBytes

A 128-bit struct that represents a UUID as raw bytes.

```
typedef struct {
    UInt8 byte0;
    UInt8 byte1;
    UInt8 byte2;
    UInt8 byte3;
    UInt8 byte4;
    UInt8 byte5;
    UInt8 byte6;
    UInt8 byte7;
    UInt8 byte8;
    UInt8 byte9;
    UInt8 byte10;
    UInt8 byte11;
    UInt8 byte12;
    UInt8 byte13;
    UInt8 byte14;
    UInt8 byte15;
} CFUUIDBytes;
```

Fields

`byte0`
The first byte.

`byte1`
The second byte.

`byte2`
The third byte.

`byte3`
The fourth byte.

`byte4`
The fifth byte.

`byte5`
The sixth byte.

`byte6`
The seventh byte.

`byte7`
The eighth byte.

`byte8`
The ninth byte.

`byte9`
The tenth byte.

`byte10`
The eleventh byte.

`byte11`
The twelfth byte.

`byte12`
The thirteenth byte.

`byte13`
The fourteenth byte.

`byte14`
The fifteenth byte.

`byte15`
The sixteenth byte.

Discussion
This structure can be obtained from a CFUUID object using the `CFUUIDGetUUIDBytes` (page 738) function. This structure is can be passed to functions that expect a raw UUID.

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

Declared In
`CFUUID.h`

CFUUIDRef

A reference to a CFUUID object.

```
typedef const struct __CFUUID *CFUUIDRef;
```

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

Declared In
`CFUUID.h`

CFWriteStream Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFStream.h

Overview

`CFWriteStream` provides an interface for writing a byte stream either synchronously or asynchronously. You can create streams that write bytes to a block of memory, a file, or a generic socket. All streams need to be opened, using CFWriteStreamOpen (page 748), before writing.

Use CFReadStream for reading byte streams, and for the functions, such as `CFStreamCreatePairWithSocketToHost` (page 865), that create socket streams).

`CFWriteStream` is "toll-free bridged" with its Cocoa Foundation counterpart, NSOutputStream. 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 `NSOutputStream *` parameter, you can pass in a `CFWriteStreamRef`, and in a function where you see a `CFWriteStreamRef` parameter, you can pass in an `NSOutputStream` instance. Note, however, that you may have either a delegate or callbacks but not both. See Interchangeable Data Types for more information on toll-free bridging.

Functions by Task

Creating a Write Stream

`CFWriteStreamCreateWithAllocatedBuffers` (page 744)
Creates a writable stream for a growable block of memory.

`CFWriteStreamCreateWithBuffer` (page 745)
Creates a writable stream for a fixed-size block of memory.

`CFWriteStreamCreateWithFile` (page 746)
Creates a writable stream for a file.

Opening and Closing a Stream

`CFWriteStreamClose` (page 743)
Closes a writable stream.

`CFWriteStreamOpen` (page 748)
Opens a stream for writing.

Writing to a Stream

`CFWriteStreamWrite` (page 751)
Writes data to a writable stream.

Scheduling a Write Stream

`CFWriteStreamScheduleWithRunLoop` (page 748)
Schedules a stream into a run loop.

`CFWriteStreamUnscheduleFromRunLoop` (page 751)
Removes a stream from a particular run loop.

Examining Stream Properties

`CFWriteStreamCanAcceptBytes` (page 743)
Returns whether a writable stream can accept new data without blocking.

`CFWriteStreamCopyProperty` (page 744)
Returns the value of a property for a stream.

`CFWriteStreamCopyError` (page 743)
Returns the error associated with a stream.

`CFWriteStreamGetError` (page 746)
Returns the error status of a stream. (Deprecated. Use `CFWriteStreamCopyError` (page 743) instead.)

`CFWriteStreamGetStatus` (page 747)
Returns the current state of a stream.

Setting Stream Properties

`CFWriteStreamSetClient` (page 749)
Assigns a client to a stream, which receives callbacks when certain events occur.

`CFWriteStreamSetProperty` (page 750)
Sets the value of a property for a stream.

Getting the CFWriteStream Type ID

`CFWriteStreamGetTypeID` (page 747)
Returns the type identifier of all CFWriteStream objects.

Functions

CFWriteStreamCanAcceptBytes

Returns whether a writable stream can accept new data without blocking.

```
Boolean CFWriteStreamCanAcceptBytes (
   CFWriteStreamRef stream
);
```

Parameters

stream
The stream to examine.

Return Value

`true` if data can be written to *stream* without blocking, `false` otherwise. If *stream* cannot tell if data can be written without actually trying to write the data, this function returns `true`.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CFStream.h`

CFWriteStreamClose

Closes a writable stream.

```
void CFWriteStreamClose (
   CFWriteStreamRef stream
);
```

Parameters

stream
The stream to close.

Discussion

This function terminates the flow of bytes and releases any system resources required by the stream. The stream is removed from any run loops in which it was scheduled. Once closed, the stream cannot be reopened.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CFFTPSample

Declared In

`CFStream.h`

CFWriteStreamCopyError

Returns the error associated with a stream.

```
CFErrorRef CFWriteStreamCopyError (
    CFReadStreamRef stream
);
```

Parameters

stream

The stream to examine.

Return Value

A CFError object that describes the current problem with stream, or `NULL` if there is no error. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CFStream.h`

CFWriteStreamCopyProperty

Returns the value of a property for a stream.

```
CFTypeRef CFWriteStreamCopyProperty (
   CFWriteStreamRef stream,
   CFStringRef propertyName
);
```

Parameters

stream

The stream to examine.

propertyName

The name of the stream property to obtain. The available properties for standard Core Foundation streams are listed in "Stream Properties." (page 872)

Return Value

The value of the property *propertyName*. Ownership follows the Create Rule.

Discussion

Each type of stream can define a set of properties that either describe or configure individual streams. A property can be any interesting information about a stream. Examples include the headers from an HTTP transmission, the expected number of bytes, file permission information, and so on. Use `CFWriteStreamSetProperty` (page 750) to modify the value of a property, although some properties are read-only.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CFStream.h`

CFWriteStreamCreateWithAllocatedBuffers

Creates a writable stream for a growable block of memory.

```
CFWriteStreamRef CFWriteStreamCreateWithAllocatedBuffers (
   CFAllocatorRef alloc,
   CFAllocatorRef bufferAllocator
);
```

Parameters

alloc

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

bufferAllocator

The allocator to use to allocate memory for the stream's memory buffers. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

Return Value
A new write stream. Ownership follows the Create Rule.

Discussion
New buffers are allocated using *bufferAllocator* as bytes are written to the stream. At any point, you can recover the bytes thus far written by asking for the property `kCFStreamPropertyDataWritten` with `CFWriteStreamCopyProperty` (page 744).

You must open the stream, using CFWriteStreamOpen (page 748), before writing to it.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CFStream.h`

CFWriteStreamCreateWithBuffer

Creates a writable stream for a fixed-size block of memory.

```
CFWriteStreamRef CFWriteStreamCreateWithBuffer (
   CFAllocatorRef alloc,
   UInt8 *buffer,
   CFIndex bufferCapacity
);
```

Parameters

alloc

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

buffer

The memory buffer into which to write data. This buffer must exist for the lifetime of the stream.

bufferCapacity

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

Return Value
A new write stream, or `NULL` on failure. Ownership follows the Create Rule.

Discussion
When *buffer* is filled after writing *bufferCapacity* bytes, the stream is exhausted and its status becomes `kCFStreamStatusAtEnd` (page 868).

You must open the stream, using CFWriteStreamOpen (page 748), before writing to it.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CFStream.h

CFWriteStreamCreateWithFile

Creates a writable stream for a file.

```
CFWriteStreamRef CFWriteStreamCreateWithFile (
   CFAllocatorRef alloc,
   CFURLRef fileURL
);
```

Parameters

alloc

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

fileURL

The URL of the file to which to write. The URL must use a file scheme.

Return Value
The new write stream, or NULL on failure. Ownership follows the Create Rule.

Discussion
The stream overwrites an existing file unless you set the kCFStreamPropertyAppendToFile to kCFBooleanTrue with CFWriteStreamSetProperty (page 750), in which case the stream appends data to the file.

You must open the stream, using CFWriteStreamOpen (page 748), before writing to it.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
CFStream.h

CFWriteStreamGetError

Returns the error status of a stream. (Deprecated. Use CFWriteStreamCopyError (page 743) instead.)

```
CFStreamError CFWriteStreamGetError (
   CFWriteStreamRef stream
);
```

Parameters

stream
The stream to examine.

Return Value
The error status of *stream* returned in a `CFStreamError` structure.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFWriteStreamGetStatus

Returns the current state of a stream.

```
CFStreamStatus CFWriteStreamGetStatus (
   CFWriteStreamRef stream
);
```

Parameters

stream
The stream to examine.

Return Value
The current state of *stream*. See `CFStreamStatus` (page 867) for the list of possible states.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CFStream.h`

CFWriteStreamGetTypeID

Returns the type identifier of all CFWriteStream objects.

```
CFTypeID CFWriteStreamGetTypeID (
   void
);
```

Return Value
The type identifier for the CFWriteStream opaque type.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFWriteStreamOpen

Opens a stream for writing.

```
Boolean CFWriteStreamOpen (
   CFWriteStreamRef stream
);
```

Parameters

stream
The stream to open.

Return Value
`true` if *stream* was successfully opened, `false` otherwise. If *stream* is not in the `kCFStreamStatusNotOpen` (page 868) state, this function returns `false`.

Discussion
Opening a stream causes it to reserve all the system resources it requires. If the stream can open in the background without blocking, this function always returns `true`. To learn when a background open operation completes, you can either schedule the stream into a run loop with `CFWriteStreamScheduleWithRunLoop` (page 748) and wait for the stream's client (set with `CFWriteStreamSetClient` (page 749)) to be notified or you can poll the stream using `CFWriteStreamGetStatus` (page 747), waiting for a status of `kCFStreamStatusOpen` (page 868) or `kCFStreamStatusError` (page 868).

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFWriteStreamScheduleWithRunLoop

Schedules a stream into a run loop.

```
void CFWriteStreamScheduleWithRunLoop (
   CFWriteStreamRef stream,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

stream
The stream to schedule.

runLoop
The run loop in which to schedule *stream*.

runLoopMode
The run loop mode of *runLoop* in which to schedule *stream*.

Discussion
After scheduling *stream* into a run loop, its client (set with `CFWriteStreamSetClient` (page 749)) is notified when various events happen with the stream, such as when it finishes opening, when it can accept new bytes, and when an error occurs. A stream can be scheduled into multiple run loops and run loop modes. Use `CFWriteStreamUnscheduleFromRunLoop` (page 751) to later remove *stream* from the run loop.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFWriteStreamSetClient

Assigns a client to a stream, which receives callbacks when certain events occur.

```
Boolean CFWriteStreamSetClient (
   CFWriteStreamRef stream,
   CFOptionFlags streamEvents,
   CFWriteStreamClientCallBack clientCB,
   CFStreamClientContext *clientContext
);
```

Parameters

stream
The stream to modify.

streamEvents
The set of events for which the client should receive callbacks. The events are listed in `CFStreamEventType` (page 871). If you pass `kCFStreamEventNone` (page 871), the current client for *stream* is removed.

clientCB
The client callback function to call when one of the events requested in *streamEvents* occurs. If `NULL`, the current client for *stream* is removed.

clientContext
A structure holding contextual information for the stream client. The function copies the information out of the structure, so the memory pointed to by *clientContext* does not need to persist beyond the function call. If `NULL`, the current client for *stream* is removed.

Return Value
`true` if the stream supports asynchronous notification, `false` otherwise.

Discussion
To avoid polling and blocking, you can register a client to hear about interesting events that occur on a stream. Only one client per stream is allowed; registering a new client replaces the previous one.

Once you have set a client, you need to schedule the stream in a run loop using `CFWriteStreamScheduleWithRunLoop` (page 748) so that the client can receive the asynchronous notifications. You can schedule each stream in multiple run loops (for example, if you are using a thread pool). It is the caller's responsibility to ensure that at least one of the scheduled run loops is being run, otherwise the callback cannot be called.

Although all Core Foundation streams currently support asynchronous notification, future stream types may not. If a stream does not support asynchronous notification, this function returns `false`. Typically, such streams never block for device I/O (for example, a stream writing to memory) and don't benefit from asynchronous notification.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

CFWriteStreamSetProperty

Sets the value of a property for a stream.

```
Boolean CFWriteStreamSetProperty (
   CFWriteStreamRef stream,
   CFStringRef propertyName,
   CFTypeRef propertyValue
);
```

Parameters

stream
The stream to modify.

propertyName
The name of the property to set. The available properties for standard Core Foundation streams are listed in "Stream Properties." (page 872)

propertyValue
The value to which to set the property *propertyName* for *stream*. The allowed data type of the value depends on the property being set.

Return Value
`true` if *stream* recognizes and accepts the given property-value pair, `false` otherwise.

Discussion
Each type of stream can define a set of properties that either describe or configure individual streams. A property can be any interesting information about a stream. Examples include the headers from an HTTP transmission, the expected number of bytes, file permission information, and so on. Properties that can be set configure the behavior of the stream and may be modifiable only at particular times, such as before the stream has been opened. (In fact, you should assume that you can set properties only before opening the stream, unless otherwise noted.) To read the value of a property use `CFWriteStreamCopyProperty` (page 744), although some properties are write-only.

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

Related Sample Code
CFFTPSample
CocoaEcho
CocoaHTTPServer
CocoaSOAP

Declared In
CFStream.h

CFWriteStreamUnscheduleFromRunLoop

Removes a stream from a particular run loop.

```
void CFWriteStreamUnscheduleFromRunLoop (
   CFWriteStreamRef stream,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

stream
The stream to remove.

runLoop
The run loop from which to remove *stream*.

runLoopMode
The run loop mode of *runLoop* from which to remove *stream*.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
CFStream.h

CFWriteStreamWrite

Writes data to a writable stream.

```
CFIndex CFWriteStreamWrite (
   CFWriteStreamRef stream,
   const UInt8 *buffer,
   CFIndex bufferLength
);
```

Parameters

stream
The stream to which to write.

buffer
The buffer holding the data to write.

bufferLength
The number of bytes from *buffer* to write.

Return Value
The number of bytes successfully written, `0` if the stream has been filled to capacity (for fixed-length streams), or `-1` if either the stream is not open or an error occurs.

Discussion
If *stream* is in the process of opening, this function waits until it has completed. If the stream is not full, this call blocks until at least one byte is written; it does not block until all the bytes in *buffer* is written. To avoid blocking, call this function only if `CFWriteStreamCanAcceptBytes` (page 743) returns `true` or after the stream's client (set with `CFWriteStreamSetClient` (page 749)) is notified of a `kCFStreamEventCanAcceptBytes` (page 871) event.

Availability
Available in Mac OS X v10.1 and later.

Related Sample Code
CFFTPSample

Declared In
`CFStream.h`

Callbacks

CFWriteStreamClientCallBack

Callback invoked when certain types of activity takes place on a writable stream.

```
typedef void (*CFWriteStreamClientCallBack) (
    CFWriteStreamRef stream,
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFWriteStreamRef stream,
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

Parameters
stream
The stream that experienced the event *eventType*.

eventType
The event that caused the callback to be called. The possible events are listed in "Stream Events." (page 871).

clientCallBackInfo
The `info` member of the `CFStreamClientContext` (page 867) structure that was used when setting the client for *stream*.

Discussion
This callback is called only for the events requested when setting the client with `CFWriteStreamSetClient` (page 749).

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

Declared In
`CFStream.h`

Data Types

CFWriteStreamRef

A reference to a writable stream object.

```
typedef struct __CFWriteStream *CFWriteStreamRef;
```

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

Declared In
`CFStream.h`

CFXMLNode Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFXMLNode.h
Companion guide	XML Programming Topics for Core Foundation

Overview

A CFXMLNode object describes an individual XML construct—like a tag, or a comment, or a string of character data. CFXMLNode is intended to be used with the CFXMLParser and CFXMLTree opaque types.

Each CFXMLNode object contains three main pieces of information—the node's type, the data string, and a pointer to an additional information data structure. A CFXMLNode object's type is one of the enumerations described in Node Type Code (page 766). The data string is always a CFString object; the meaning of the string is dependent on the node's type. The format of the additional data is also dependent on the node's type; in general, there is a custom structure for each type that requires additional data. See Node Type Code (page 766) for the mapping from a node type to meaning of the data string, and structure of the additional information. Note that these structures are versioned and may change as the parser changes. The current version can always be identified by the `kCFXMLNodeCurrentVersion` (page 766) constant; earlier versions can be identified and used by passing earlier values for the version number (although the older structures would have been removed from the header).

You create a CFXMLNode object using one of the create or copy functions. Use the `CFXMLNodeGetTypeCode` (page 758), `CFXMLNodeGetString` (page 757), and `CFXMLNodeGetInfoPtr` (page 757) functions to get the node type, data string, and additional information respectively. Use the `CFXMLNodeGetVersion` (page 758) function to get a node's version number.

Functions

CFXMLNodeCreate

Creates a new CFXMLNode.

```
CFXMLNodeRef CFXMLNodeCreate (
   CFAllocatorRef alloc,
   CFXMLNodeTypeCode xmlType,
   CFStringRef dataString,
   const void *additionalInfoPtr,
   CFIndex version
);
```

Parameters

alloc

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

xmlType

Type identifier code for the XML structure you want this node to describe.

dataString

The XML data.

additionalInfoPtr

A pointer to a structure containing additional information about the XML data.

version

The version number of the CFXMLNode object you want to create. Pass one of the pre-defined constants, typically `kCFXMLNodeCurrentVersion` (page 766).

Return Value
A new CFXMLNode object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

CFXMLNodeCreateCopy

Creates a copy of a CFXMLNode object.

```
CFXMLNodeRef CFXMLNodeCreateCopy (
   CFAllocatorRef alloc,
   CFXMLNodeRef origNode
);
```

Parameters

alloc

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

origNode

The node to copy. Do not pass `NULL`.

Return Value
A new CFXMLNode object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

CFXMLNodeGetInfoPtr

Returns the additional information pointer of a CFXMLNode object.

```
const void * CFXMLNodeGetInfoPtr (
   CFXMLNodeRef node
);
```

Parameters

node
The CFXMLNode object to examine.

Return Value
A pointer to a structure containing additional information. The CFXMLNode version together with the node's type determines the expected structure. See Node Type Code (page 766) for information about the possible structures returned. If the returned value is a Core Foundation object, ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

CFXMLNodeGetString

Returns the data string from a CFXMLNode.

```
CFStringRef CFXMLNodeGetString (
   CFXMLNodeRef node
);
```

Parameters

node
The CFXMLNode object to examine.

Return Value
The data string from *node*. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

CFXMLNodeGetTypeCode

Returns the XML structure type code for a CFXMLNode object.

```
CFXMLNodeTypeCode CFXMLNodeGetTypeCode (
   CFXMLNodeRef node
);
```

Parameters

node

The CFXMLNode object to examine.

Return Value

The type code for *node*.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFXMLNode.h

CFXMLNodeGetTypeID

Returns the type identifier code for the CFXMLNode opaque type.

```
CFTypeID CFXMLNodeGetTypeID (
   void
);
```

Return Value

The type identifier for the CFXMLNode opaque type.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFXMLNode.h

CFXMLNodeGetVersion

Returns the version number for a CFXMLNode object.

```
CFIndex CFXMLNodeGetVersion (
   CFXMLNodeRef node
);
```

Parameters

node

The CFXMLNode object to examine.

Return Value

The version number of *node*.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

Data Types

CFXMLAttributeDeclarationInfo

Contains information about an element attribute definition.

```
struct CFXMLAttributeDeclarationInfo {
    CFStringRef attributeName;
    CFStringRef typeString;
    CFStringRef defaultString;
};
typedef struct CFXMLAttributeDeclarationInfo CFXMLAttributeDeclarationInfo;
```

Fields

`attributeName`
The name of the attribute.

`typeString`
Describes the declaration of a single attribute.

`defaultString`
The attribute's default value.

Discussion
This structure is part of the definition of the `CFXMLAttributeListDeclarationInfo` (page 759) structure.

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

Declared In
`CFXMLNode.h`

CFXMLAttributeListDeclarationInfo

Contains a list of the attributes associated with an element.

```
struct CFXMLAttributeListDeclarationInfo {
    CFIndex numberOfAttributes;
    CFXMLAttributeDeclarationInfo *attributes;
};
typedef struct CFXMLAttributeListDeclarationInfo  CFXMLAttributeListDeclarationInfo;
```

Fields

`numberOfAttributes`
The number of attributes in the array.

`attributes`
A C array of attributes.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters an attribute declaration in the DTD. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain the pointer to this structure.

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

Declared In
`CFXMLNode.h`

CFXMLDocumentInfo

Contains the source URL and text encoding information for the XML document.

```
struct CFXMLDocumentInfo {
    CFURLRef sourceURL;
    CFStringEncoding encoding;
};
typedef struct CFXMLDocumentInfo CFXMLDocumentInfo;
```

Fields
`sourceURL`
The source URL of the XML document.

`encoding`
The text encoding of the XML document.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters the XML declaration. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain the pointer.

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

Declared In
`CFXMLNode.h`

CFXMLDocumentTypeInfo

Contains the external ID of the DTD.

```
struct CFXMLDocumentTypeInfo {
    CFXMLExternalID externalID;
};
typedef struct CFXMLDocumentTypeInfo CFXMLDocumentTypeInfo;
```

Fields
`externalID`
The external ID of the DTD.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters the beginning of the DTD. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain a pointer to this structure.

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

Declared In
`CFXMLNode.h`

CFXMLElementInfo

Contains a list of element attributes packaged as CFDictionary key/value pairs.

```
struct CFXMLElementInfo {
    CFDictionaryRef attributes;
    CFArrayRef attributeOrder;
    Boolean isEmpty;
};
typedef struct CFXMLElementInfo CFXMLElementInfo;
```

Fields

`attributes`
The dictionary of attribute values.

`attributeOrder`
An array specifying the order in which the attributes appeared in the XML document.

`isEmpty`
A flag indicating whether the element was expressed in closed form.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters an element containing attributes. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain the pointer.

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

Declared In
`CFXMLNode.h`

CFXMLElementTypeDeclarationInfo

Contains a description of the element type.

```
struct CFXMLElementTypeDeclarationInfo {
    CFStringRef contentDescription;
};
typedef struct CFXMLElementTypeDeclarationInfo CFXMLElementTypeDeclarationInfo;
```

Fields

`contentDescription`
A textual description of the element type.

Discussion
A pointer to this structure is included in the CFXMLNode passed to your application when the parser encounters and element type declaration. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain a pointer to this structure.

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

Declared In
`CFXMLNode.h`

CFXMLEntityInfo

Contains information describing an XML entity.

```
struct CFXMLEntityInfo {
    CFXMLEntityTypeCode entityType;
    CFStringRef replacementText;
    CFXMLExternalID entityID;
    CFStringRef notationName;
};
typedef struct CFXMLEntityInfo CFXMLEntityInfo;
```

Fields

`entityType`
The entity type code.

`replacementText`
`NULL` if `entityType` is external or unparsed, otherwise the text that the entity should be replaced with.

`entityID`
`entityID.systemID` will be `NULL` if `entityType` is internal.

`notationName`
`NULL` if `entityType` is parsed.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters an entity declaration. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain a pointer to this structure.

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

Declared In
`CFXMLNode.h`

CFXMLEntityReferenceInfo

Contains information describing an XML entity reference.

```
struct CFXMLEntityReferenceInfo {
    CFXMLEntityTypeCode entityType;
};
typedef struct CFXMLEntityReferenceInfo CFXMLEntityReferenceInfo;
```

Fields

`entityType`

The entity type code.

Discussion

A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters an entity reference. Use the `CFXMLNodeGetInfoPtr` (page 757) function to obtain the pointer.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLNode.h`

CFXMLExternalID

Contains the system and public IDs for an external entity reference.

```
struct CFXMLExternalID {
    CFURLRef systemID;
    CFStringRef publicID;
};
typedef struct CFXMLExternalID CFXMLExternalID;
```

Fields

`systemID`

The systemID URL.

`publicID`

The publicID string.

Discussion

This structure is part of the definition of the `CFXMLDocumentTypeInfo` (page 760), `CFXMLNotationInfo` (page 764), and `CFXMLEntityInfo` (page 762) structures.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLNode.h`

CFXMLNodeRef

A reference to a CFXMLNode object.

```
typedef const struct __CFXMLNode *CFXMLNodeRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
CFXMLNode.h

CFXMLNotationInfo

Contains the external ID of the notation.

```
struct CFXMLNotationInfo {
    CFXMLExternalID externalID;
};
typedef struct CFXMLNotationInfo CFXMLNotationInfo;
```

Fields
externalID
The external ID of the notation.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters a notation element. Use the CFXMLNodeGetInfoPtr (page 757) function to obtain a pointer to this structure.

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

Declared In
CFXMLNode.h

CFXMLProcessingInstructionInfo

Contains the text of the processing instruction.

```
struct CFXMLProcessingInstructionInfo {
    CFStringRef dataString;
};
typedef struct CFXMLProcessingInstructionInfo CFXMLProcessingInstructionInfo;
```

Fields
dataString
The text of the processing instruction.

Discussion
A pointer to this structure is included in the CFXMLNode object passed to your application when the parser encounters a processing instruction. Use the CFXMLNodeGetInfoPtr (page 757) function to obtain the pointer.

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

Declared In
CFXMLNode.h

Constants

Entity Type Code

The entity type identification codes that the parser uses to describe XML entities.

```
enum CFXMLEntityTypeCode {
    kCFXMLEntityTypeParameter = 0,
    kCFXMLEntityTypeParsedInternal = 1,
    kCFXMLEntityTypeParsedExternal = 2,
    kCFXMLEntityTypeUnparsed = 3,
    kCFXMLEntityTypeCharacter = 4
};
typedef enum CFXMLEntityTypeCode CFXMLEntityTypeCode;
```

Constants

`kCFXMLEntityTypeParameter`
Implies a parsed, internal entity.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLEntityTypeParsedInternal`
Indicates a parsed, internal entity.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLEntityTypeParsedExternal`
Indicates a parsed, external entity.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLEntityTypeUnparsed`
Indicates an unparsed entity.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLEntityTypeCharacter`
Indicates a character entity type.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

Discussion

These codes are used with the `CFXMLEntityInfo` (page 762) and `CFXMLEntityReferenceInfo` (page 762) structures.

Node Current Version

The version of a CFXMLNode object.

```
enum {
    kCFXMLNodeCurrentVersion = 1
};
```

Constants

`kCFXMLNodeCurrentVersion`

The current version of CFXMLNode objects.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

Node Type Code

The various XML data type identification codes that the parser uses to describe XML structures.

```
enum CFXMLNodeTypeCode {
    kCFXMLNodeTypeDocument = 1,
    kCFXMLNodeTypeElement = 2,
    kCFXMLNodeTypeAttribute = 3,
    kCFXMLNodeTypeProcessingInstruction = 4,
    kCFXMLNodeTypeComment = 5,
    kCFXMLNodeTypeText = 6,
    kCFXMLNodeTypeCDATASection = 7,
    kCFXMLNodeTypeDocumentFragment = 8,
    kCFXMLNodeTypeEntity = 9,
    kCFXMLNodeTypeEntityReference = 10,
    kCFXMLNodeTypeDocumentType = 11,
    kCFXMLNodeTypeWhitespace = 12,
    kCFXMLNodeTypeNotation = 13,
    kCFXMLNodeTypeElementTypeDeclaration = 14,
    kCFXMLNodeTypeAttributeListDeclaration = 15
};
typedef enum CFXMLNodeTypeCode CFXMLNodeTypeCode;
```

Constants

`kCFXMLNodeTypeDocument`

Indicates a document where the data string is `NULL` and the additional information is a pointer to a `CFXMLDocumentInfo` (page 760) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeElement`

Indicates an element where the data string is the name of the tag and the additional information is a pointer to a `CFXMLElementInfo` (page 761) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeAttribute`

Currently not used.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeProcessingInstruction`
Indicates a processing instruction where the data string is the name of the target and the additional information is a pointer to a `CFXMLProcessingInstructionInfo` (page 764) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeComment`
Indicates a comment section where the data string is the text of the comment and the additional information is `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeText`
Indicates a text section where the data string is the text's contents and the additional information is `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeCDATASection`
Indicates a CDATA section where the data string is the text of the CDATA and the additional information is `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeDocumentFragment`
Currently not used.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeEntity`
Indicates an entity where the data string is the name of the entity and the additional information is a pointer to a `CFXMLEntityInfo` (page 762) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeEntityReference`
Indicates an entity reference where the data string is the name of the referenced entity and the additional information is a pointer to a `CFXMLEntityReferenceInfo` (page 762) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeDocumentType`
Indicates a document type where the data string is the name given to the top-level element and the additional information is a pointer to a `CFXMLDocumentTypeInfo` (page 760) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeWhitespace`
Indicates white space where the data string is the text of the white space and the additional information is `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeNotation`
Indicates a notation where the data string is the notation name and the additional information is a pointer to a `CFXMLNotationInfo` (page 764) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeElementTypeDeclaration`
Indicates an element type declaration where the data string is the tag name and the additional information is a pointer to a `CFXMLElementTypeDeclarationInfo` (page 761) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

`kCFXMLNodeTypeAttributeListDeclaration`
Indicates an attribute list declaration where the data string is the tag name and the additional information is a pointer to a `CFXMLAttributeListDeclarationInfo` (page 759) structure.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLNode.h`.

Discussion
When the parser encounters a new XML structure, its data type and contents are placed in a CFXMLNode object.

CFXMLParser Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFXMLParser.h
Companion guide	XML Programming Topics for Core Foundation

Overview

CFXMLParser provides an XML parser you can use to find and extract data in XML documents. You can use a high-level interface to load an XML document into a Core Foundation collection object. A low-level callback-based interface allows you to perform any action you wish on an XML structured type when it is detected by the parser. This opaque type is relevant for applications that need information about an XML document's structure or content.

Functions

CFXMLParserAbort

Causes a parser to abort with the given error code and description.

```
void CFXMLParserAbort (
    CFXMLParserRef parser,
    CFXMLParserStatusCode errorCode,
    CFStringRef errorDescription
);
```

Parameters

parser

The parser to abort.

errorCode

The error code to return to the parser.

errorDescription

The error description string to return to the parser. This value may not be `NULL`.

Discussion

This function cannot be called asynchronously. In other words, it must be called from within a parser callback function.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserCopyErrorDescription

Returns the user-readable description of the current error condition.

```
CFStringRef CFXMLParserCopyErrorDescription (
    CFXMLParserRef parser
);
```

Parameters

parser
The XML parser to examine.

Return Value
A user-readable description of the current error condition, or `NULL` if no error occurred. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserCreate

Creates a new XML parser for the specified XML data.

```
CFXMLParserRef CFXMLParserCreate (
    CFAllocatorRef allocator,
    CFDataRef xmlData,
    CFURLRef dataSource,
    CFOptionFlags parseOptions,
    CFIndex versionOfNodes,
    CFXMLParserCallBacks *callBacks,
    CFXMLParserContext *context
);
```

Parameters

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

xmlData
The XML data to parse. Do not pass `NULL`.

dataSource
The URL from which the XML data was obtained. The URL is used to resolve any relative references found in XML Data. Pass `NULL` if a valid URL is unavailable.

parseOptions
Flags which control how the XML data will be parsed. See Parsing Options (page 786) for the list of available options.

versionOfNodes
Determines which version of CFXMLNode objects are produced by the parser.

callBacks
Callbacks called by the parser as the XML is processed. The callbacks are called as each XML tag is encountered, when an external entity needs to be resolved, and when an error occurs. See `CFXMLParserCallBacks` (page 782) and the individual callbacks for more details. Do not pass `NULL`.

context
Determines what, if any, information pointer is passed to the callbacks as the parse progresses; *context* may be `NULL`.

Return Value
The newly created parser. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserCreateWithDataFromURL

Creates a new XML parser for the specified XML data at the specified URL.

```
CFXMLParserRef CFXMLParserCreateWithDataFromURL (
    CFAllocatorRef allocator,
    CFURLRef dataSource,
    CFOptionFlags parseOptions,
    CFIndex versionOfNodes,
    CFXMLParserCallBacks *callBacks,
    CFXMLParserContext *context
);
```

Parameters

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

dataSource
The URL from which to load the XML data. The URL is used to resolve any relative references found in XML Data. It must be a valid CFURL object; `NULL` is an unacceptable value.

parseOptions
Flags which control how the XML data will be parsed. See Parsing Options (page 786) for the list of available options.

versionOfNodes
Determines which version of CFXMLNode objects are produced by the parser.

callBacks
Callbacks called by the parser as the XML is processed. The callbacks are called as each XML tag is encountered, when an external entity needs to be resolved, and when an error occurs. See `CFXMLParserCallBacks` (page 782) and the individual callbacks for more details. Do not pass `NULL`.

context
Determines what, if any, information pointer is passed to the callbacks as the parse progresses; may be `NULL`.

Return Value
The newly created parser. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetCallBacks

Returns the callbacks associated with an XML parser when it was created.

```
void CFXMLParserGetCallBacks (
    CFXMLParserRef parser,
    CFXMLParserCallBacks *callBacks
);
```

Parameters

parser
The XML parser to examine.

callBacks
On return, contains the callbacks for *parser*.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetContext

Returns the context for an XML parser.

```
void CFXMLParserGetContext (
    CFXMLParserRef parser,
    CFXMLParserContext *context
);
```

Parameters

parser
The XML parser to examine.

context
On return, a pointer to the context structure for *parser*.

Discussion
If you set a context for the parser, it will be passed to you as a parameter in each of the parser callback functions. The context data structure is application defined and associated with a parser using one of the `CFXMLParserCreate...` functions.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetDocument

Returns the top-most object returned by the create XML structure callback.

```
void *CFXMLParserGetDocument (
    CFXMLParserRef parser
);
```

Parameters
parser
The XML parser to examine.

Return Value
The top-most object returned by the `createXMLStructure` field in the `CFXMLParserCallBacks` (page 782) structure. If the returned value is a Core Foundation object, ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetLineNumber

Returns the line number of the current parse location.

```
CFIndex CFXMLParserGetLineNumber (
    CFXMLParserRef parser
);
```

Parameters
parser
The XML parser to examine.

Return Value
The line number of the current location.

Discussion
This function is typically used in conjunction with the `CFXMLParserHandleErrorCallBack` (page 779) function so that error location information can be reported.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetLocation

Returns the character index of the current parse location.

```
CFIndex CFXMLParserGetLocation (
    CFXMLParserRef parser
);
```

Parameters

parser
The XML parser to examine.

Return Value
The character index of the current parse location.

Discussion
This function is typically used in conjunction with the `CFXMLParserHandleErrorCallBack` (page 779) function so that error location information can be reported.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetSourceURL

Returns the URL for the XML data being parsed.

```
CFURLRef CFXMLParserGetSourceURL (
    CFXMLParserRef parser
);
```

Parameters

parser
The XML parser to examine.

Return Value
The URL for the XML document being parsed. Ownership follows the Get Rule.

Availability
Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetStatusCode

Returns a numeric code indicating the current status of the parser.

```
CFXMLParserStatusCode CFXMLParserGetStatusCode (
    CFXMLParserRef parser
);
```

Parameters

parser
The XML parser to examine.

Return Value
A status code indicating the current parser. See Parser Status Codes (page 784) for a list of possible status codes.

Discussion
If an error has occurred, the code for the last error is returned. If no error has occurred, a status code is returned.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserGetTypeID

Returns the type identifier for the CFXMLParser opaque type.

```
CFTypeID CFXMLParserGetTypeID ();
```

Return Value
The type identifier for the CFXMLParser opaque type.

Availability
Available in CarbonLib v1.3 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLParserParse

Begins a parse of the XML data that was associated with the parser when it was created.

```
Boolean CFXMLParserParse (
    CFXMLParserRef parser
);
```

Parameters

parser

The XML parser to start.

Return Value

`true` if the parse was successful, `false` otherwise.

Discussion

Upon success, use the `CFXMLParserGetDocument` (page 773) function to get the product of the parse. Upon failure, use the `CFXMLParserGetContext` (page 772) or `CFXMLParserCopyErrorDescription` (page 770) functions to get information about the error. It is an error to call the `CFXMLParserParse` (page 775) function while a parse is already underway.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

Callbacks

CFXMLParserAddChildCallBack

Callback function invoked by the parser to notify your application of parent/child relationships between XML structures.

```
typedef void (*CFXMLParserAddChildCallBack) (
    CFXMLParserRef parser,
    void *parent,
    void *child,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFXMLParserRef parser,
    void *parent,
    void *child,
    void *info
);
```

Parameters

parser

The CFXMLParser object making the callback.

parent

The program-defined value representing the XML element to whom *child* is being added. This value was returned by the CFXMLParserCreateXMLStructureCallBack (page 778) callback when this element's open tag was detected.

child

The program-defined value representing the XML element that is being added to *parent*. This value was returned by the CFXMLParserCreateXMLStructureCallBack (page 778) callback when this element's open tag was detected.

info

The program-defined context data you specified in the CFXMLParserContext (page 783) structure when creating the parser.

Discussion
If the CFXMLParserCreateXMLStructureCallBack (page 778) function returns NULL for a given structure, that structure is omitted entirely, and this callback will *not* be called for either a NULL child or parent.

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

Declared In
CFXMLParser.h

CFXMLParserCopyDescriptionCallBack

Callback function invoked by the parser when handling the information pointer.

```
typedef CFStringRef (*CFXMLParserCopyDescriptionCallBack) (
    const void *info
);
```

If you name your function MyCallBack, you would declare it like this:

```
CFStringRef MyCallBack (
    const void *info
);
```

Parameters

info

The program-defined context data you specified in the CFXMLParserContext (page 783) structure when creating the parser.

Return Value
A textual description of *info*. The caller is responsible for releasing this object.

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

Declared In
CFXMLParser.h

CFXMLParserCreateXMLStructureCallBack

Callback function invoked when the parser encounters an XML open tag.

```
typedef void *(*CFXMLParserCreateXMLStructureCallBack) (
    CFXMLParserRef parser,
    CFXMLNodeRef nodeDesc,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    CFXMLParserRef parser,
    CFXMLNodeRef nodeDesc,
    void *info
);
```

Parameters

parser

The CFXMLParser object making the callback.

nodeDesc

The CFXMLNode object that represents the XML structure encountered.

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

Return Value

A program-defined value representing the new XML element or `NULL` to indicate that the given structure should be skipped. This value is passed to the other callbacks.

Discussion

If `NULL` is returned for a given structure, only minimal parsing is done for that structure (enough to correctly determine its end, and to extract any data necessary for the remainder of the parse, such as Entity definitions). This callback (or any of the tree-creation callbacks) will not be called for any children of the skipped structure. The only exception is that the top-most element will always be reported even if `NULL` was returned for the document as a whole. For performance reasons, the node passed to this callback cannot be safely retained by the client; the node as a whole must be copied (using the `CFXMLNodeCreateCopy` (page 756) function), or its contents must be extracted and copied. You are required to implement this callback for the parser to operate.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserEndXMLStructureCallBack

Callback function invoked by the parser to notify your application that an XML structure (and all its children) have been completely parsed.

```
typedef void (*CFXMLParserEndXMLStructureCallBack) (
    CFXMLParserRef parser,
    void *xmlType,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    CFXMLParserRef parser,
    void *xmlType,
    void *info
);
```

Parameters

parser

The CFXMLParser object making the callback.

xmlType

The program-defined value representing the XML element whose end tag has been detected. This value was returned by the `CFXMLParserCreateXMLStructureCallBack` (page 778) callback.

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

Discussion

As elements are encountered, this callback is called first, then the `CFXMLParserAddChildCallBack` (page 776) callback to add the new structure to its parent, then the `CFXMLParserAddChildCallBack` (page 776) callback (potentially several times) to add the new structure's children to it, and then finally the `CFXMLParserEndXMLStructureCallBack` (page 778) callback to show that the structure has been fully parsed.This callback is optional.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserHandleErrorCallBack

Callback function invoked by the parser to notify your application that an error has occurred.

```
typedef Boolean (*CFXMLParserHandleErrorCallBack) (
    CFXMLParserRef parser,
    CFXMLParserStatusCode error,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
    CFXMLParserRef parser,
    CFXMLParserStatusCode error,
    void *info
```

```
);
```

Parameters

parser

A CFXMLParser object making the callback.

error

A status code describing the error.

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

Return Value

`true` if the parser should continue parsing the XML, `false` if the parser should stop.

Discussion

If this callback is not defined, the parser will silently attempt to recover. Otherwise, this callback may return `false` to force the parser to stop. If this callback returns `true`, the parser will attempt to recover (fatal errors will still cause the parse to abort immediately). This callback is optional.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserReleaseCallBack

Callback function invoked by the parser when it wants to release a reference to the information pointer.

```
typedef void (*CFXMLParserReleaseCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    const void *info
);
```

Parameters

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserResolveExternalEntityCallBack

Callback function invoked by the parser to notify your application that an external entity has been referenced.

```
typedef CFDataRef (*CFXMLParserResolveExternalEntityCallBack) (
    CFXMLParserRef parser,
    CFXMLExternalID *extID,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFDataRef MyCallBack (
    CFXMLParserRef parser,
    CFXMLExternalID *extID,
    void *info
);
```

Parameters

parser

The CFXMLParser object making the callback.

extID

The identifier for the external entity.

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

Return Value

The external entity or `NULL` if it should not be resolved.

Discussion

If this callback is not defined, the parser uses its internal routines to try and resolve the entity. Otherwise, if this callback returns `NULL`, a place holder for the external entity is inserted into the tree. In this manner, the parser's client can prevent any external network or file accesses. This callback is optional.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserRetainCallBack

Callback function invoked by the parser when it needs another reference to the information pointer.

```
typedef const void *(*CFXMLParserRetainCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
const void *MyCallBack (
    const void *info
);
```

Parameters

info

The program-defined context data you specified in the `CFXMLParserContext` (page 783) structure when creating the parser.

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

Declared In
`CFXMLParser.h`

Data Types

CFXMLParserCallBacks

Contains version information and function pointers to callbacks needed when parsing XML.

```
struct CFXMLParserCallBacks {
    CFIndex version;
    CFXMLParserCreateXMLStructureCallBack createXMLStructure;
    CFXMLParserAddChildCallBack addChild;
    CFXMLParserEndXMLStructureCallBack endXMLStructure;
    CFXMLParserResolveExternalEntityCallBack resolveExternalEntity;
    CFXMLParserHandleErrorCallBack handleError;
};
typedef struct CFXMLParserCallBacks CFXMLParserCallBacks;
```

Fields

`version`

Version number. Must be `0`.

`createXMLStructure`

Called when an XML structure is created.

`addChild`

Called when a child is added.

`endXMLStructure`

Called when an XML structure has ended.

`resolveExternalEntity`

Called when an external entity needs to be resolved.

`handleError`

Called when a parse error needs to be handled.

Discussion
This structure is passed to one of the `CFXMLParserCreate...` functions. Only the *`createXMLStructure`*, *`addChild`*, and *`endXMLStructure`* fields are required. Set the others to `NULL` if you don't wish to implement them.

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

Declared In
`CFXMLParser.h`

CFXMLParserContext

Contains version information and function pointers to callbacks used when handling a program-defined context.

```
struct CFXMLParserContext {
    CFIndex version;
    void *info;
    CFXMLParserRetainCallBack retain;
    CFXMLParserReleaseCallBack release;
    CFXMLParserCopyDescriptionCallBack copyDescription;
};
typedef struct CFXMLParserContext CFXMLParserContext;
```

Fields

`version`
Version number of this structure. Must be 0.

`info`
An arbitrary program-defined value passed to all the callbacks in this structure and in the `CFXMLParserCallBacks` (page 782) structure.

`retain`
A retain callback for your program-defined context data. Optional.

`release`
A release callback for your program-defined context data. Optional.

`copyDescription`
A copy description callback for your program-defined context data. Optional.

Discussion

You can associate a context with a parser when the parser is created. The context can be anything you wish and will be passed as a parameter to all of the XML parser callbacks.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

CFXMLParserRef

A reference to an XML parser object.

```
typedef struct __CFXMLParser *CFXMLParserRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFXMLParser.h`

Constants

Parser Status Codes

The various status and error flags that can be returned by the parser.

```
enum CFXMLParserStatusCode {
    kCFXMLStatusParseNotBegun = -2,
    kCFXMLStatusParseInProgress = -1,
    kCFXMLStatusParseSuccessful = 0,
    kCFXMLErrorUnexpectedEOF = 1,
    kCFXMLErrorUnknownEncoding = 2,
    kCFXMLErrorEncodingConversionFailure = 3,
    kCFXMLErrorMalformedProcessingInstruction = 4,
    kCFXMLErrorMalformedDTD = 5,
    kCFXMLErrorMalformedName = 6,
    kCFXMLErrorMalformedCDSect = 7,
    kCFXMLErrorMalformedCloseTag = 8,
    kCFXMLErrorMalformedStartTag = 9,
    kCFXMLErrorMalformedDocument = 10,
    kCFXMLErrorElementlessDocument = 11,
    kCFXMLErrorMalformedComment = 12,
    kCFXMLErrorMalformedCharacterReference = 13,
    kCFXMLErrorMalformedParsedCharacterData = 14,
    kCFXMLErrorNoData = 15
};
typedef enum CFXMLParserStatusCode CFXMLParserStatusCode;
```

Constants

`kCFXMLStatusParseNotBegun`
Indicates the parser has not begun.
Available in Mac OS X v10.0 and later.
Declared in `CFXMLParser.h`.

`kCFXMLStatusParseInProgress`
Indicates the parser is in progress.
Available in Mac OS X v10.0 and later.
Declared in `CFXMLParser.h`.

`kCFXMLStatusParseSuccessful`
Indicates the parser was successful.
Available in Mac OS X v10.0 and later.
Declared in `CFXMLParser.h`.

`kCFXMLErrorUnexpectedEOF`
Indicates an unexpected EOF occurred.
Available in Mac OS X v10.0 and later.
Declared in `CFXMLParser.h`.

`kCFXMLErrorUnknownEncoding`
Indicates an unknown encoding error.
Available in Mac OS X v10.0 and later.
Declared in `CFXMLParser.h`.

`kCFXMLErrorEncodingConversionFailure`
Indicates an encoding conversion error.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedProcessingInstruction`
Indicates a malformed processing instruction.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedDTD`
Indicates a malformed DTD.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedName`
Indicates a malformed name.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedCDSect`
Indicates a malformed CDATA section.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedCloseTag`
Indicates a malformed close tag.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedStartTag`
Indicates a malformed start tag.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedDocument`
Indicates a malformed document.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorElementlessDocument`
Indicates a document containing no elements.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedComment`
Indicates a malformed comment.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedCharacterReference`
Indicates a malformed character reference.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorMalformedParsedCharacterData`
Indicates malformed character data.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLErrorNoData`
Indicates a no data error.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

Discussion
Parser status is determined by calling the `CFXMLParserGetStatusCode` (page 775) function. The parser reports errors to your application by invoking the `CFXMLParserHandleErrorCallBack` (page 779) function.

Parsing Options

Options you can use to control the parser's treatment of an XML document.

```
enum CFXMLParserOptions {
    kCFXMLParserValidateDocument = (1 << 0),
    kCFXMLParserSkipMetaData = (1 << 1),
    kCFXMLParserReplacePhysicalEntities = (1 << 2),
    kCFXMLParserSkipWhitespace = (1 << 3),
    kCFXMLParserResolveExternalEntities = (1 << 4),
    kCFXMLParserAddImpliedAttributes = (1 << 5),
    kCFXMLParserAllOptions = 0x00FFFFFF,
    kCFXMLParserNoOptions = 0
};
typedef enum CFXMLParserOptions CFXMLParserOptions;
```

Constants
`kCFXMLParserValidateDocument`
Validates the document against its grammar from the DTD, reporting any errors. Currently not supported.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLParserSkipMetaData`
Silently skip over metadata constructs (the DTD and comments).

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLParserReplacePhysicalEntities`
Replaces declared entities like `<`. Note that other than the 5 predefined entities (`lt`, `gt`, `quot`, `amp`, `apos`), these must be defined in the DTD. Currently not supported.

Available in Mac OS X v10.0 and later.

Declared in `CFXMLParser.h`.

`kCFXMLParserSkipWhitespace`
: Skip over all whitespace that does not abut non-whitespace character data. In other words, given "`<foo> <bar> blah </bar></foo>`," the whitespace between foo's open tag and bar's open tag would be suppressed, but the whitespace around `blah` would be preserved.

  Available in Mac OS X v10.0 and later.

  Declared in `CFXMLParser.h`.

`kCFXMLParserResolveExternalEntities`
: Resolves all external entities.

  Available in Mac OS X v10.0 and later.

  Declared in `CFXMLParser.h`.

`kCFXMLParserAddImpliedAttributes`
: Where the DTD specifies implied attribute-value pairs for a particular element, add those pairs to any occurrences of the element in the element tree. Currently not supported.

  Available in Mac OS X v10.0 and later.

  Declared in `CFXMLParser.h`.

`kCFXMLParserAllOptions`
: Makes the parser do the most work, returning only the pure elementtree.

  Available in Mac OS X v10.0 and later.

  Declared in `CFXMLParser.h`.

`kCFXMLParserNoOptions`
: Leaves the XML as "intact" as possible (reports all structures; performs no replacements).

  Available in Mac OS X v10.0 and later.

  Declared in `CFXMLParser.h`.

Discussion
These are the various options you use to configure the parser. An option flag of 0 (`kCFXMLParserNoOptions` (page 787)) leaves the XML as "intact" as possible (reports all structures; performs no replacements). Hence, to make the parser do the most work, returning only the pure element tree, set the option flag to `kCFXMLParserAllOptions` (page 787).

CFXMLTree Reference

Derived From:	*CFTree Reference* : *CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFXMLParser.h CFXMLNode.h
Companion guide	XML Programming Topics for Core Foundation

Overview

A CFXMLTree object is simply a CFTree object whose context data is known to be a CFXMLNode object. CFXMLTree is derived from CFTree—you can pass CFXMLTree objects in all the CFTree functions. As such, a CFXMLTree object can be used to represent an entire XML document; the CFTree object provides the tree structure of the document, while the CFXMLNode objects identify and describe the nodes of the tree. An XML document can be parsed to a CFXMLTree object, and a CFXMLTree object can generate the data for the equivalent XML document. This opaque type is expected to be used in conjunction with CFXMLParser and CFXMLNode objects.

Functions

CFXMLCreateStringByEscapingEntities

Given a CFString object containing XML source with unescaped entities, returns a string with specified XML entities escaped.

```
CFStringRef CFXMLCreateStringByEscapingEntities(
    CFAllocatorRef allocator,
    CFStringRef string,
    CFDictionaryRef entitiesDictionary,
);
```

Parameters

allocator

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

string

Any CFString object that may contain XML source. This function translates any substring that is mapped to an entity in *entitiesDictionary* to the specified entity.

entitiesDictionary
Specifies the entities to be replaced. Dictionary keys should be the entity names (for example, "para" for ¶), and the values should be CFString objects containing the expansion. Pass `NULL` to indicate no entities other than the standard five.

Return Value
A CFString object derived from *string* with substrings identified in *entitiesDictionary* escaped to their corresponding entities. Ownership follows the Create Rule.

Discussion
The standard five predefined entities are automatically supported.

As an example of using this function, say you apply this function to string "Refer to ¶ 5 of the contract" with a key of "para" mapped to "¶" in *entitiesDictionary*. The resulting string is "Refer to ¶ 5 of the contract".

Note: Currently, only the standard predefined entities are supported; passing `NULL` for *entitiesDictionary* is sufficient.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFXMLParser.h`

CFXMLCreateStringByUnescapingEntities

Given a CFString object containing XML source with escaped entities, returns a string with specified XML entities unescaped.

```
CFStringRef CFXMLCreateStringByUnescapingEntities(
    CFAllocatorRef allocator,
    CFStringRef string,
    CFDictionaryRef entitiesDictionary,
);
```

Parameters

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

string
Any CFString object that may contain XML source. This function translates any entity that is mapped to an substring in *entitiesDictionary* to the specified substring.

entitiesDictionary
Specifies the entities to be replaced. Dictionary keys should be the entity names (for example, "para" for ¶), and the values should be CFString objects containing the expansion. Pass `NULL` to indicate no entities other than the standard five.

Return Value
A CFString object derived from *string* with entities identified in *entitiesDictionary* unescaped to their corresponding substrings. Ownership follows the Create Rule.

Discussion
The standard five predefined entities are automatically supported.

As an example of using this function, say you apply this function to string "Refer to ¶ 5 of the contract" with a key of "para" mapped to "¶" in *entitiesDictionary*. The resulting string is "Refer to ¶ 5 of the contract".

Note: Currently, only the standard predefined entities are supported; passing `NULL` for *entitiesDictionary* is sufficient.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFXMLParser.h`

CFXMLTreeCreateFromData

Parses the given XML data and returns the resulting CFXMLTree object.

```
CFXMLTreeRef CFXMLTreeCreateFromData (
    CFAllocatorRef allocator,
    CFDataRef xmlData,
    CFURLRef dataSource,
    CFOptionFlags parseOptions,
    CFIndex versionOfNodes
);
```

Parameters

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

xmlData
The XML data you wish to parse.

dataSource
The URL from which the XML data was obtained. The URL is used to resolve any relative references found in *xmlData*. Pass `NULL` if a valid URL is unavailable.

parseOptions
Flags which control how the XML data will be parsed. See `Parsing Options` (page 786) for the list of available options.

versionOfNodes
Determines which version of CFXMLNode objects are produced by the parser.

Return Value
A new CFXMLTree object containing the data from the specified XML document. Ownership follows the Create Rule.

Discussion
This function represents the high-level interface to the XML parser. This single function creates a parser for the specified XML data using the specified options. The parser creates and returns a CFXMLTree object that you can examine and modify with the CFTree functions or obtain the node using the `CFXMLTreeGetNode` (page 795) function and examine its attributes using CFXMLNode functions.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLTreeCreateFromDataWithError

Parses the given XML data and returns the resulting CFXMLTree object and any error information.

```
CFXMLTreeRef CFXMLTreeCreateFromDataWithError (
    CFAllocatorRef allocator,
    CFDataRef xmlData,
    CFURLRef dataSource,
    CFOptionFlags parseOptions,
    CFIndex versionOfNodes
    CFDictionaryRef *errorDict
);
```

Parameters

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

xmlData
The XML data you wish to parse.

dataSource
The URL from which the XML data was obtained. The URL is used to resolve any relative references found in *xmlData*. Pass `NULL` if a valid URL is unavailable.

parseOptions
Flags which control how the XML data will be parsed. See `Parsing Options` (page 786) for the list of available options.

versionOfNodes
Determines which version of CFXMLNode objects are produced by the parser. The current version is 1.

errorDict
Upon return, if an error occurs contains a CFDictionary object that describes the error. If no errors occur, this parameter is not changed. Pass `NULL` if you don't want error information. See "Error Dictionary Keys" (page 795) for a description of the key-value pairs in this dictionary. Ownership follows the Create Rule.

Return Value
A new CFXMLTree object containing the data from the specified XML document. Ownership follows the Create Rule.

Discussion
Use this function instead of `CFXMLTreeCreateFromData` (page 791) if you need access to XML parsing errors.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CFXMLParser.h`

CFXMLTreeCreateWithDataFromURL

Creates a new CFXMLTree object by loading the data to be parsed directly from a data source.

```
CFXMLTreeRef CFXMLTreeCreateWithDataFromURL (
    CFAllocatorRef allocator,
    CFURLRef dataSource,
    CFOptionFlags parseOptions,
    CFIndex versionOfNodes
);
```

Parameters

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

dataSource
The URL from which the XML data is obtained. The URL is used to resolve any relative references found in XML Data. Pass `NULL` if a valid URL is unavailable.

parseOptions
Flags which control how the XML data will be parsed. See `Parsing Options` (page 786) for the list of available options.

versionOfNodes
Determines which version of CFXMLNode objects are produced by the parser.

Return Value
A new CFXMLTree object containing the data from the specified XML data source. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLTreeCreateWithNode

Creates a childless, parentless CFXMLTree object node for a CFXMLNode object.

```
CFXMLTreeRef CFXMLTreeCreateWithNode (
    CFAllocatorRef allocator,
    CFXMLNodeRef node
);
```

Parameters

allocator

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

node

The CFXMLNode object to use when creating the new CFXMLTree object.

Return Value
A CFXMLTree object. Ownership follows the Create Rule.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLNode.h`

CFXMLTreeCreateXMLData

Generates an XML document from a CFXMLTree object which is ready to be written to permanent storage.

```
CFDataRef CFXMLTreeCreateXMLData (
    CFAllocatorRef allocator,
    CFXMLTreeRef xmlTree
);
```

Parameters

allocator

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

xmlTree

The CFXMLTree object you wish to convert to an XML document.

Return Value
The XML data. Ownership follows the Create Rule.

Discussion
This function will *not* regenerate entity references replaced at the parse time (except those required for syntactic correctness). If you need this you must manually walk the tree and re-insert any entity references that should appear in the final output file.

Availability
Available in CarbonLib v1.1 and later.
Available in Mac OS X v10.0 and later.

Declared In
`CFXMLParser.h`

CFXMLTreeGetNode

Returns the node of a CFXMLTree object.

```
CFXMLNodeRef CFXMLTreeGetNode (
    CFXMLTreeRef xmlTree
);
```

Parameters

xmlTree

The CFXMLTree object whose node you wish to obtain.

Return Value

The node of *xmlTree*. Ownership follows the Get Rule.

Availability

Available in CarbonLib v1.1 and later.

Available in Mac OS X v10.0 and later.

Declared In

CFXMLNode.h

Data Types

CFXMLTreeRef

A reference to a CFXMLTree object.

```
typedef CFTreeRef CFXMLTreeRef;
```

Discussion

When using the high-level parser API, XML data is parsed to a special CFTree object which is simply a CFXMLTree object with known contexts and callbacks. The nodes of a CFXMLTree may be queried using the basic CFTree functions (to report on the structure of the tree itself), or via the functions here (to report on the XML contents of the nodes).

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFXMLNode.h

Constants

Error Dictionary Keys

The keys used in an error dictionary returned by some functions to provide more information about XML parse errors.

```
const CFStringRef kCFXMLTreeErrorDescription;
const CFStringRef kCFXMLTreeErrorLineNumber;
const CFStringRef kCFXMLTreeErrorLocation;
const CFStringRef kCFXMLTreeErrorStatusCode;
```

Constants

`kCFXMLTreeErrorDescription`

Dictionary key whose value is a CFString containing a readable description of the error.

Available in Mac OS X v10.3 and later.

Declared in `CFXMLParser.h`.

`kCFXMLTreeErrorLineNumber`

Dictionary key whose value is a CFNumber containing the line number where the error was detected. This may not be the line number where the actual XML error is located.

Available in Mac OS X v10.3 and later.

Declared in `CFXMLParser.h`.

`kCFXMLTreeErrorLocation`

Dictionary key whose value is a CFNumber containing the byte location where the error was detected.

Available in Mac OS X v10.3 and later.

Declared in `CFXMLParser.h`.

`kCFXMLTreeErrorStatusCode`

Dictionary key whose value is a CFNumber containing the error status code. See *CFXMLParser Reference* for possible status code values.

Available in Mac OS X v10.3 and later.

Declared in `CFXMLParser.h`.

Discussion

These keys are used in the error dictionary returned by the `CFXMLTreeCreateFromDataWithError` (page 792) function.

Availability

Available in Mac OS X v10.3 and later.

Managers

Base Utilities Reference

Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFBase.h
Companion guide	Core Foundation Design Concepts

Overview

Core Foundation defines a number of miscellaneous symbols that are either used by many different opaque types, such as `CFIndex` (page 801), or apply to Core Foundation as a whole, such as `kCFCoreFoundationVersionNumber` (page 803). These symbols are collected together and documented here.

Functions

CFRangeMake

Declares and initializes a `CFRange` structure.

```
CFRange CFRangeMake (
   CFIndex loc,
   CFIndex len
);
```

Parameters

loc

The starting location of the range.

len

The length of the range.

Return Value

An initialized structure of type `CFRange` (page 802).

Discussion

This is an in-line convenience function for creating initialized `CFRange` (page 802) structures.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

CoreTextTest
DockBrowser
ImageClient
MoreSCF

Declared In
`CFBase.h`

Callbacks

CFComparatorFunction

Callback function that compares two values. You provide a pointer to this callback in certain Core Foundation sorting functions.

```
typedef CFComparisonResult (*CFComparatorFunction) (
    const void *val1,
    const void *val2,
    void *context
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFComparisonResult MyCallBack (
    const void *val1,
    const void *val2,
    void *context
);
```

Parameters

val1

The first value to compare.

val2

The second value to compare.

context

An untyped pointer to the context of the evaluation.

The meaning of this value and its use are defined by each comparator function. This value is usually passed to a sort function, such as `CFArraySortValues` (page 290), which then passes it, unchanged, to the comparator function.

Return Value
A `CFComparisonResult` value that indicates whether the *val1* is equal to, less than, or greater than *val2*. See "Comparison Results" (page 802) for a list of possible values.

Discussion
If you need to sort the elements in a collection using special criteria, you can implement a comparator function with the signature defined by this prototype. You pass a pointer to this function in one of the "sort" functions, such as CFArray's `CFArraySortValues` (page 290).

You can also pass pointers to standard Core Foundation comparator functions such as `CFStringCompare` (page 542) and `CFDateCompare` (page 179).

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

Declared In
`CFBase.h`

Data Types

CFIndex

An integer type used throughout Core Foundation in several programmatic roles: as an array index and for count, size, and length parameters and return values.

```
typedef signed long CFIndex;
```

Discussion
Core Foundation types as `CFIndex` all parameters and return values that might grow over time as the processor's address size changes. On architectures where pointer sizes are a different size (say, 64 bits) `CFIndex` might be declared to be also 64 bits, independent of the size of `int`.

If you type your own variables that interact with Core Foundation as `CFIndex`, your code will have a higher degree of source compatibility in the future.

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

Declared In
`CFBase.h`

CFOptionFlags

A bitfield used for passing special allocation and other requests into Core Foundation functions.

```
typedef UInt32 CFOptionFlags;
```

Discussion
The flag bits are specific to particular opaque types and functions in Core Foundation.

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

Declared In
`CFBase.h`

CFRange

A structure representing a range of sequential items in a container, such as characters in a buffer or elements in a collection.

```
struct CFRange {
    CFIndex location;
    CFIndex length;
};
typedef struct CFRange CFRange;
```

Fields

`location`

An integer representing the starting location of the range.

`length`

An integer representing the number of items in the range.

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

Declared In
`CFBase.h`

Constants

Comparison Results

Constants returned by comparison functions, indicating whether a value is equal to, less than, or greater than another value.

```
enum CFComparisonResult {
    kCFCompareLessThan = -1,
    kCFCompareEqualTo = 0,
    kCFCompareGreaterThan = 1
};
typedef enum CFComparisonResult CFComparisonResult;
```

Constants

`kCFCompareLessThan`

Returned by a comparison function if the first value is less than the second value.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFCompareEqualTo`

Returned by a comparison function if the first value is equal to the second value.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFCompareGreaterThan`

Returned by a comparison function if the first value is greater than the second value.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

Value Not Found

Special value returned when a Core Foundation function cannot locate a requested value.

```
enum {
    kCFNotFound = -1
};
```

Constants

`kCFNotFound`

A constant that indicates that a search operation did not succeed in locating the target value.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

Current Framework Version Number

Current version number of the Core Foundation framework.

```
double kCFCoreFoundationVersionNumber;
```

Constants

`kCFCoreFoundationVersionNumber`

The current version of the Core Foundation framework. Compare this value to the values in "Framework Version Numbers" (page 803). Although this variable was added to the CFBase.h header file in Mac OS X v10.1, it was available and can be used in Mac OS X v10.0.

Available in Mac OS X v10.1 and later.

Declared in `CFBase.h`.

Framework Version Numbers

Version numbers of the Core Foundation framework.

```
#define kCFCoreFoundationVersionNumber10_0    196.40
#define kCFCoreFoundationVersionNumber10_0_3    196.50
#define kCFCoreFoundationVersionNumber10_1    226.00
#define kCFCoreFoundationVersionNumber10_1_1    226.00
/* Note the next three do not follow the usual numbering policy
   from the base release */
#define kCFCoreFoundationVersionNumber10_1_2    227.20
#define kCFCoreFoundationVersionNumber10_1_3    227.20
#define kCFCoreFoundationVersionNumber10_1_4    227.30
#define kCFCoreFoundationVersionNumber10_2    263.00
#define kCFCoreFoundationVersionNumber10_2_1    263.10
#define kCFCoreFoundationVersionNumber10_2_2    263.10
#define kCFCoreFoundationVersionNumber10_2_3    263.30
#define kCFCoreFoundationVersionNumber10_2_4    263.30
#define kCFCoreFoundationVersionNumber10_2_5    263.50
#define kCFCoreFoundationVersionNumber10_2_6    263.50
#define kCFCoreFoundationVersionNumber10_2_7    263.50
#define kCFCoreFoundationVersionNumber10_2_8    263.50
#define kCFCoreFoundationVersionNumber10_3    299.00
#define kCFCoreFoundationVersionNumber10_3_1    299.00
#define kCFCoreFoundationVersionNumber10_3_2    299.00
#define kCFCoreFoundationVersionNumber10_3_3    299.30
#define kCFCoreFoundationVersionNumber10_3_4    299.31
#define kCFCoreFoundationVersionNumber10_3_5    299.31
#define kCFCoreFoundationVersionNumber10_3_6    299.32
#define kCFCoreFoundationVersionNumber10_3_7    299.33
#define kCFCoreFoundationVersionNumber10_3_8    299.33
#define kCFCoreFoundationVersionNumber10_3_9    299.35
#define kCFCoreFoundationVersionNumber10_4    368.00
#define kCFCoreFoundationVersionNumber10_4_1    368.10
#define kCFCoreFoundationVersionNumber10_4_2    368.11
#define kCFCoreFoundationVersionNumber10_4_3    368.18
#define kCFCoreFoundationVersionNumber10_4_4_Intel    368.26
#define kCFCoreFoundationVersionNumber10_4_4_PowerPC    368.25
#define kCFCoreFoundationVersionNumber10_4_5_Intel    368.26
#define kCFCoreFoundationVersionNumber10_4_5_PowerPC    368.25
#define kCFCoreFoundationVersionNumber10_4_6_Intel    368.26
#define kCFCoreFoundationVersionNumber10_4_6_PowerPC    368.25
#define kCFCoreFoundationVersionNumber10_4_7    368.27
#define kCFCoreFoundationVersionNumber10_4_8    368.27
#define kCFCoreFoundationVersionNumber10_4_9    368.28
#define kCFCoreFoundationVersionNumber10_4_10    368.28
#define kCFCoreFoundationVersionNumber10_4_11    368.31
#define kCFCoreFoundationVersionNumber10_5    476.00
#define kCFCoreFoundationVersionNumber10_5_1    476.00
#define kCFCoreFoundationVersionNumber10_5_2    476.10
#define kCFCoreFoundationVersionNumber10_5_3    476.13
#define kCFCoreFoundationVersionNumber10_5_4    476.14
#define kCFCoreFoundationVersionNumber10_5_5    476.15
#define kCFCoreFoundationVersionNumber10_5_6    476.17
```

```
#define kCFCoreFoundationVersionNumber_iOS_2_0    478.23
#define kCFCoreFoundationVersionNumber_iOS_2_1 478.26
#define kCFCoreFoundationVersionNumber_iOS_2_2 478.29
```

Constants

`kCFCoreFoundationVersionNumber10_0`
The Core Foundation framework version in Mac OS X version 10.0.
Available in Mac OS X v10.1 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_0_3`
The Core Foundation framework version in Mac OS X version 10.0.3.
Available in Mac OS X v10.1 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_1`
The Core Foundation framework version in Mac OS X version 10.1.
Available in Mac OS X v10.2 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_1_1`
The Core Foundation framework version in Mac OS X version 10.1.1.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_1_2`
The Core Foundation framework version in Mac OS X version 10.1.2.
Available in Mac OS X v10.3 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_1_3`
The Core Foundation framework version in Mac OS X version 10.1.3.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_1_4`
The Core Foundation framework version in Mac OS X version 10.1.4.
Available in Mac OS X v10.3 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2`
The Core Foundation framework version in Mac OS X version 10.2.
Available in Mac OS X v10.3 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_1`
The Core Foundation framework version in Mac OS X version 10.2.1.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_2`
The Core Foundation framework version in Mac OS X version 10.2.2.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_3`
The Core Foundation framework version in Mac OS X version 10.2.3.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_4`
The Core Foundation framework version in Mac OS X version 10.2.4.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_5`
The Core Foundation framework version in Mac OS X version 10.2.5.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_6`
The Core Foundation framework version in Mac OS X version 10.2.6.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_7`
The Core Foundation framework version in Mac OS X version 10.2.7.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_2_8`
The Core Foundation framework version in Mac OS X version 10.2.8.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3`
The Core Foundation framework version in Mac OS X version 10.3.
Available in Mac OS X v10.4 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_1`
The Core Foundation framework version in Mac OS X version 10.3.1.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_2`
The Core Foundation framework version in Mac OS X version 10.3.2.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_3`
The Core Foundation framework version in Mac OS X version 10.3.3.
Available in Mac OS X v10.4 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_4`
The Core Foundation framework version in Mac OS X version 10.3.4.
Available in Mac OS X v10.4 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_5`
The Core Foundation framework version in Mac OS X version 10.3.5.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_6`
The Core Foundation framework version in Mac OS X version 10.3.6.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_7`
The Core Foundation framework version in Mac OS X version 10.3.7.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_8`
The Core Foundation framework version in Mac OS X version 10.3.8.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_3_9`
The Core Foundation framework version in Mac OS X version 10.3.9.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4`
The Core Foundation framework version in Mac OS X version 10.4.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_1`
The Core Foundation framework version in Mac OS X version 10.4.1.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_2`
The Core Foundation framework version in Mac OS X version 10.4.2.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_3`
The Core Foundation framework version in Mac OS X version 10.4.3.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_4_Intel`
The Core Foundation framework version in Mac OS X version 10.4.4 on Intel-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_4_PowerPC`
The Core Foundation framework version in Mac OS X version 10.4.4 on PowerPC-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_5_Intel`
The Core Foundation framework version in Mac OS X version 10.4.5 on Intel-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_5_PowerPC`
The Core Foundation framework version in Mac OS X version 10.4.5 on PowerPC-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_6_Intel`
The Core Foundation framework version in Mac OS X version 10.4.6 on Intel-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_6_PowerPC`
The Core Foundation framework version in Mac OS X version 10.4.6 on PowerPC-based Macintosh computers.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_7`
The Core Foundation framework version in Mac OS X version 10.4.7.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_8`
The Core Foundation framework version in Mac OS X version 10.4.8.

Available in Mac OS X v10.5 and later.

Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_9`
The Core Foundation framework version in Mac OS X version 10.4.9.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_10`
The Core Foundation framework version in Mac OS X version 10.4.10.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_4_11`
The Core Foundation framework version in Mac OS X version 10.4.11.
Available in Mac OS X v10.5 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5`
The Core Foundation framework version in Mac OS X version 10.5.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_1`
The Core Foundation framework version in Mac OS X version 10.5.1.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_2`
The Core Foundation framework version in Mac OS X version 10.5.2.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_3`
The Core Foundation framework version in Mac OS X version 10.5.3.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_4`
The Core Foundation framework version in Mac OS X version 10.5.4.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_5`
The Core Foundation framework version in Mac OS X version 10.5.5.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber10_5_6`
The Core Foundation framework version in Mac OS X version 10.5.6.
Available in Mac OS X v10.6 and later.
Declared in `CFBase.h`.

`kCFCoreFoundationVersionNumber_iOS_2_0`
The Core Foundation framework version in iOS version 2.0.

`kCFCoreFoundationVersionNumber_iOS_2_1`
The Core Foundation framework version in iOS version 2.1.

`kCFCoreFoundationVersionNumber_iOS_2_2`
The Core Foundation framework version in iOS version 2.2.

Discussion
Compare these values to the value of the `kCFCoreFoundationVersionNumber` (page 803) variable to identify on which version of the Core Foundation framework your code is executing.

Byte-Order Utilities Reference

Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFByteOrder.h
Companion guide	Memory Management Programming Guide for Core Foundation

Overview

When handling binary data transmitted or shared across platforms, you need be concerned with how each platform stores numerical values. A platform stores values either in big-endian or little-endian format. On big-endian machines, such as PowerPC machines, values are stored with the most-significant bytes first in memory; on little-endian machines, such as Pentium machines, values are stored with the least-significant bytes first. A multibyte value transmitted to a platform with a different format will be misinterpreted if it is not converted properly by one of the computers.

You identify the native format of the current platform using the `CFByteOrderGetCurrent` (page 811) function. Use functions such as `CFSwapInt32BigToHost` (page 817) and `CFConvertFloat32HostToSwapped` (page 812) to convert values between different byte order formats.

Functions

CFByteOrderGetCurrent

Returns the byte order of the current computer.

```
CFByteOrder CFByteOrderGetCurrent (
   void
);
```

Return Value
The byte order of the current computer. See "Byte Order Flags" (page 822) for the list of possible return values.

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

Declared In
`CFByteOrder.h`

CFConvertDoubleHostToSwapped

Converts a 64-bit double from the host's native byte order to a platform-independent format.

```
CFSwappedFloat64 CFConvertDoubleHostToSwapped (
   double arg
);
```

Parameters

arg

The real value to convert.

Return Value

A structure holding the real value in a canonical byte order.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFByteOrder.h

CFConvertDoubleSwappedToHost

Converts a 64-bit double from a platform-independent format to the host's native byte order.

```
double CFConvertDoubleSwappedToHost (
   CFSwappedFloat64 arg
);
```

Parameters

arg

A structure holding the real value to convert.

Return Value

The real value in the host's native format.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFByteOrder.h

CFConvertFloat32HostToSwapped

Converts a 32-bit float from the host's native byte order to a platform-independent format.

```
CFSwappedFloat32 CFConvertFloat32HostToSwapped (
   Float32 arg
);
```

Parameters

arg

The real value to convert.

Return Value

A structure holding the real value in a canonical byte order.

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

Declared In
`CFByteOrder.h`

CFConvertFloat32SwappedToHost

Converts a 32-bit float from a platform-independent format to the host's native byte order.

```
Float32 CFConvertFloat32SwappedToHost (
   CFSwappedFloat32 arg
);
```

Parameters

arg

A structure holding the real value to convert.

Return Value
The real value in the host's native format.

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

Declared In
`CFByteOrder.h`

CFConvertFloat64HostToSwapped

Converts a 64-bit float from the host's native byte order to a platform-independent format.

```
CFSwappedFloat64 CFConvertFloat64HostToSwapped (
   Float64 arg
);
```

Parameters

arg

The real value to convert.

Return Value
A structure holding the real value in a canonical byte order.

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

Declared In
`CFByteOrder.h`

CFConvertFloat64SwappedToHost

Converts a 64-bit float from a platform-independent format to the host's native byte order.

```
Float64 CFConvertFloat64SwappedToHost (
   CFSwappedFloat64 arg
);
```

Parameters

arg

A structure holding the real value to convert.

Return Value
The real value in the host's native format.

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

Declared In
CFByteOrder.h

CFConvertFloatHostToSwapped

Converts a 32-bit float from the host's native byte order to a platform-independent format.

```
CFSwappedFloat32 CFConvertFloatHostToSwapped (
   float arg
);
```

Parameters

arg

The real value to convert.

Return Value
A structure holding the real value in a canonical byte order.

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

Declared In
CFByteOrder.h

CFConvertFloatSwappedToHost

Converts a 32-bit float from a platform-independent format to the host's native byte order.

```
float CFConvertFloatSwappedToHost (
   CFSwappedFloat32 arg
);
```

Parameters

arg

A structure holding the real value to convert.

Return Value
The real value in the host's native format.

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

Declared In
CFByteOrder.h

CFSwapInt16

Swaps the bytes of a 16-bit integer.

```
uint16_t CFSwapInt16 (
   uint16_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped.

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

Declared In
CFByteOrder.h

CFSwapInt16BigToHost

Converts a 16-bit integer from big-endian format to the host's native byte order.

```
uint16_t CFSwapInt16BigToHost (
   uint16_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

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

Related Sample Code
AudioCodecSDK

Declared In
CFByteOrder.h

CFSwapInt16HostToBig

Converts a 16-bit integer from the host's native byte order to big-endian format.

```
uint16_t CFSwapInt16HostToBig (
   uint16_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioCodecSDK

Declared In

CFByteOrder.h

CFSwapInt16HostToLittle

Converts a 16-bit integer from the host's native byte order to little-endian format.

```
uint16_t CFSwapInt16HostToLittle (
   uint16_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

PTPPassThrough

Declared In

CFByteOrder.h

CFSwapInt16LittleToHost

Converts a 16-bit integer from little-endian format to the host's native byte order.

```
uint16_t CFSwapInt16LittleToHost (
   uint16_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioCodecSDK

PTPPassThrough

Declared In

CFByteOrder.h

CFSwapInt32

Swaps the bytes of a 32-bit integer.

```
uint32_t CFSwapInt32 (
   uint32_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFByteOrder.h

CFSwapInt32BigToHost

Converts a 32-bit integer from big-endian format to the host's native byte order.

```
uint32_t CFSwapInt32BigToHost (
   uint32_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

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

Related Sample Code
AudioCodecSDK
AudioQueueTools
From A View to A Movie

Declared In
CFByteOrder.h

CFSwapInt32HostToBig

Converts a 32-bit integer from the host's native byte order to big-endian format.

```
uint32_t CFSwapInt32HostToBig (
   uint32_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

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

Related Sample Code
RecordAudioToFile

Declared In
CFByteOrder.h

CFSwapInt32HostToLittle

Converts a 32-bit integer from the host's native byte order to little-endian format.

```
uint32_t CFSwapInt32HostToLittle (
   uint32_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

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

Related Sample Code
PTPPassThrough

Declared In
`CFByteOrder.h`

CFSwapInt32LittleToHost

Converts a 32-bit integer from little-endian format to the host's native byte order.

```
uint32_t CFSwapInt32LittleToHost (
   uint32_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

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

Related Sample Code
AudioCodecSDK
PTPPassThrough

Declared In
`CFByteOrder.h`

CFSwapInt64

Swaps the bytes of a 64-bit integer.

```
uint64_t CFSwapInt64 (
   uint64_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped.

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

Declared In
`CFByteOrder.h`

CFSwapInt64BigToHost

Converts a 64-bit integer from big-endian format to the host's native byte order.

```
uint64_t CFSwapInt64BigToHost (
   uint64_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFByteOrder.h

CFSwapInt64HostToBig

Converts a 64-bit integer from the host's native byte order to big-endian format.

```
uint64_t CFSwapInt64HostToBig (
   uint64_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is big-endian, this function returns *arg* unchanged.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CFByteOrder.h

CFSwapInt64HostToLittle

Converts a 64-bit integer from the host's native byte order to little-endian format.

```
uint64_t CFSwapInt64HostToLittle (
   uint64_t arg
);
```

Parameters

arg

The integer whose bytes should be swapped.

Return Value

The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

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

Related Sample Code
PTPPassThrough

Declared In
`CFByteOrder.h`

CFSwapInt64LittleToHost

Converts a 64-bit integer from little-endian format to the host's native byte order.

```
uint64_t CFSwapInt64LittleToHost (
   uint64_t arg
);
```

Parameters

arg
The integer whose bytes should be swapped.

Return Value
The integer with its bytes swapped. If the host is little-endian, this function returns *arg* unchanged.

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

Related Sample Code
PTPPassThrough

Declared In
`CFByteOrder.h`

Data Types

CFSwappedFloat32

Structure holding a 32-bit float value in a platform-independent byte order.

```
struct CFSwappedFloat32 {
    uint32_t v;
};
typedef struct CFSwappedFloat32 CFSwappedFloat32;
```

Fields

`v`
A 32-bit float value stored with a platform-independent byte order.

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

Declared In
CFByteOrder.h

CFSwappedFloat64

Structure holding a 64-bit float value in a platform-independent byte order.

```
struct CFSwappedFloat64 {
    uint64_t v;
};
typedef struct CFSwappedFloat64 CFSwappedFloat64;
```

Fields
v
A 64-bit float value stored with a platform-independent byte order.

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

Declared In
CFByteOrder.h

Constants

Byte Order Flags

Flags that identify byte order.

```
enum __CFByteOrder {
    CFByteOrderUnknown,
    CFByteOrderLittleEndian,
    CFByteOrderBigEndian
};
typedef enum __CFByteOrder CFByteOrder;
```

Constants
CFByteOrderUnknown
The byte order is unknown.
Available in Mac OS X v10.0 and later.
Declared in CFByteOrder.h.

CFByteOrderLittleEndian
Multi-byte values are stored with the least-significant bytes stored first. Pentium CPUs are little endian.
Available in Mac OS X v10.0 and later.
Declared in CFByteOrder.h.

CFByteOrderBigEndian
Multi-byte values are stored with the most-significant bytes stored first. PowerPC CPUs are big endian.
Available in Mac OS X v10.0 and later.
Declared in CFByteOrder.h.

Core Foundation URL Access Utilities Reference

Framework: CoreFoundation/CoreFoundation.h

Declared in CFURLAccess.h

Overview

Core Foundation URL Access Utilities give you convenient system-independent methods of creating, reading, updating, or deleting a URL resource.

Given a CFURL object that holds either a file or http URL, you can read the resource's data with the `CFURLCreateDataAndPropertiesFromResource` (page 823) function. You can write data to the URL resource, possibly creating a new file, with the `CFURLWriteDataAndPropertiesToResource` (page 826) function. Finally, you can destroy, or delete, the resource pointed to by the URL with the `CFURLDestroyResource` (page 825) function.

Functions

CFURLCreateDataAndPropertiesFromResource

Loads the data and properties referred to by a given URL.

```
Boolean CFURLCreateDataAndPropertiesFromResource (
   CFAllocatorRef alloc,
   CFURLRef url,
   CFDataRef *resourceData,
   CFDictionaryRef *properties,
   CFArrayRef desiredProperties,
   SInt32 *errorCode
);
```

Parameters

allocator

The allocator to use to allocate memory for the new `CFData` and `CFDictionary` objects returned in *resourceData* and *properties*. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

url

The URL referring to the data and/or properties you wish to load.

resourceData

On return, contains a `CFData` object containing the data referred to by *url*. Ownership follows the Create Rule.

properties
On return, a pointer to a `CFDictionary` object containing the resource properties referred to by *url*. Ownership follows the Create Rule.

desiredProperties
A list of the properties you wish to obtain and return in *properties*. See "File URL Properties" (page 828) and "HTTP URL Properties" (page 829) for the list of available properties.

errorCode
`0` if successful, otherwise an error code indicating the nature of the problem. See "Error Codes" (page 827) for a list of possible error codes.

Return Value
`true` if successful, `false` otherwise.

Discussion
If you are interested in loading only the resource data or the resource's properties, pass `NULL` for the one you don't want. If *properties* is non-`NULL` and *desiredProperties* is `NULL` then all properties are fetched. Note that as much work as possible is done even if `false` is returned. For instance, if one property is not available, the others are fetched anyway. This function is intended for convenience, not performance.

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

Related Sample Code
CFPrefTopScores
GLUT
HID Dumper
HID Utilities Source
SeeMyFriends

Declared In
`CFURLAccess.h`

CFURLCreatePropertyFromResource

Returns a given property specified by a given URL and property string.

```
CFTypeRef CFURLCreatePropertyFromResource (
   CFAllocatorRef alloc,
   CFURLRef url,
   CFStringRef property,
   SInt32 *errorCode
);
```

Parameters

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

url
The `CFURL` object referring to the resource whose properties are loaded.

property
The name of the property you wish to load. Pass one of the provided string constants indicating the property. See "File URL Properties" (page 828) and "HTTP URL Properties" (page 829) for the list of available properties.

errorCode
On return, `0` if successful, otherwise an error code indicating the nature of the problem. See "Error Codes" (page 827) for a list of possible error codes.

Return Value
If successful, the requested property as a `CFType` object, `NULL` otherwise. Ownership follows the Create Rule.

Discussion
This is a convenience function for retrieving individual property values which calls through to `CFURLCreateDataAndPropertiesFromResource` (page 823).

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

Declared In
`CFURLAccess.h`

CFURLDestroyResource

Destroys a resource indicated by a given URL.

```
Boolean CFURLDestroyResource (
   CFURLRef url,
   SInt32 *errorCode
);
```

Parameters

url
The `CFURL` object of the resource to destroy.

errorCode
On return, `0` if successful, otherwise an error code indicating the nature of the problem. See "Error Codes" (page 827) for a list of possible error codes.

Return Value
`true` if successful, `false` otherwise.

Discussion
If *url* uses an http scheme, an http `DELETE` request is sent to the resource. If *url* uses a file scheme, then:

- if the reference is a file, the file is deleted;
- if the reference is a directory and the directory is empty, the directory is deleted;
- if the reference is a directory and the directory is not empty, the function returns `false` and *errorCode* contains `kCFURLUnknownError`.

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

Declared In
`CFURLAccess.h`

CFURLWriteDataAndPropertiesToResource

Writes the given data and properties to a given URL.

```
Boolean CFURLWriteDataAndPropertiesToResource (
   CFURLRef url,
   CFDataRef dataToWrite,
   CFDictionaryRef propertiesToWrite,
   SInt32 *errorCode
);
```

Parameters

url

The resource to write.

dataToWrite

The data to write. Pass `NULL` to write only properties.

propertiesToWrite

The properties to write. Pass `NULL` to write only data. See "File URL Properties" (page 828) and "HTTP URL Properties" (page 829) for the list of available properties.

errorCode

Upon return, `0` if successful, otherwise contains an error code indicating the nature of the problem. See "Error Codes" (page 827) for a list of possible error codes.

Return Value

`true` if successful, `false` otherwise.

Discussion

Properties not present in *propertiesToWrite* are left unchanged, hence if *propertiesToWrite* is `NULL` or empty, the URL's properties are not changed at all.

If *url* uses a file scheme and it references a file, the contents of *dataToWrite* are written to the referenced file, overwriting any preexisting data, and the file's properties are modified according to *propertiesToWrite*. If the file does not exist, but all intermediate directories along the path do already exist, the file is created (otherwise it is not).

If *url* uses a file scheme and it references a directory (the last path character is "`/`"), the contents of *dataToWrite* are ignored, but if the parameter value is not `NULL`—and all intermediate directories along the path do already exist—a new directory is created (otherwise it is not).

If *url* uses an http scheme, an http `PUT` request is sent to the resource with *propertiesToWrite* as the header fields and *dataToWrite* as the data.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPrefTopScores
HID Calibrator
HID Config Save
HID Utilities
SeeMyFriends

Declared In

`CFURLAccess.h`

Constants

Error Codes

`CFURL` error codes.

```
enum CFURLError {
    kCFURLUnknownError = -10,
    kCFURLUnknownSchemeError = -11,
    kCFURLResourceNotFoundError = -12,
    kCFURLResourceAccessViolationError = -13,
    kCFURLRemoteHostUnavailableError = -14,
    kCFURLImproperArgumentsError = -15,
    kCFURLUnknownPropertyKeyError = -16,
    kCFURLPropertyKeyUnavailableError = -17,
    kCFURLTimeoutError = -18
};
typedef enum CFURLError CFURLError;
```

Constants

`kCFURLUnknownError`
Indicates an unknown error.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLUnknownSchemeError`
Indicates that the scheme is not recognized.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLResourceNotFoundError`
Indicates a resource was not found.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLResourceAccessViolationError`
Indicates an error in accessing a resource.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLRemoteHostUnavailableError`
Indicates a remote host is unavailable.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLImproperArgumentsError`
Indicates one or more arguments are improper.
Available in Mac OS X v10.0 and later.
Declared in `CFURLAccess.h`.

`kCFURLUnknownPropertyKeyError`
Indicates a property key is unknown.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLPropertyKeyUnavailableError`
Indicates a property key was unavailable.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLTimeoutError`
Indicates a timeout.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CFURLAccess.h`

File URL Properties

Properties for file URL resources.

```
const CFStringRef kCFURLFileExists;
const CFStringRef kCFURLFileDirectoryContents;
const CFStringRef kCFURLFileLength;
const CFStringRef kCFURLFileLastModificationTime;
const CFStringRef kCFURLFilePOSIXMode;
const CFStringRef kCFURLFileOwnerID;
```

Constants
`kCFURLFileExists`
A `CFBoolean` object indicating whether the file referred to by a URL exists.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLFileDirectoryContents`
A `CFArray` object holding `CFURL` objects for the contents of a directory referred to by a URL.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLFileLength`
A `CFNumber` object holding the file's length in bytes.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLFileLastModificationTime`
A `CFDate` object holding the file's modification time.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLFilePOSIXMode`
: A `CFNumber` holding the file's POSIX mode as given in `/usr/include/sys/stat.h`.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLFileOwnerID`
: A `CFNumber` holding the file owner's UID.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CFURLAccess.h`

HTTP URL Properties

Properties for HTTP URL resources.

```
const CFStringRef kCFURLHTTPStatusCode;
const CFStringRef kCFURLHTTPStatusLine;
```

Constants
`kCFURLHTTPStatusCode`
: A `CFNumber` object holding the status code of an HTTP request.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

`kCFURLHTTPStatusLine`
: A `CFString` object holding the status line of an HTTP request.

Available in Mac OS X v10.0 and later.

Declared in `CFURLAccess.h`.

Discussion
In addition to the above properties, each field within an HTTP request or response header is itself a property. You can specify a header field by using the field name as the property name.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CFURLAccess.h`

Preferences Utilities Reference

Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFPreferences.h
Companion guide	Preferences Programming Topics for Core Foundation

Overview

Core Foundation provides a simple, standard way to manage user (and application) preferences. Core Foundation stores preferences as key-value pairs that are assigned a scope using a combination of user name, application ID, and host (computer) names. This makes it possible to save and retrieve preferences that apply to different classes of users. Core Foundation preferences is useful to all applications that support user preferences. Note that modification of some preferences domains (those not belonging to the "Current User") requires Admin privileges—see *Authorization Services Programming Guide* for information on how to gain suitable privileges.

Unlike some other Core Foundation types, CFPreferences is not toll-free bridged to its corresponding Cocoa Foundation framework class (NSUserDefaults).

Functions by Task

Several functions return a preference value as a Core Foundation property list object. You can use the function `CFGetTypeID` (page 655) to determine the value's type. For more information about property lists, see *Property List Programming Topics for Core Foundation*.

Getting Preference Values

`CFPreferencesCopyAppValue` (page 835)
Obtains a preference value for the specified key and application.

`CFPreferencesCopyKeyList` (page 836)
Constructs and returns the list of all keys set in the specified domain.

`CFPreferencesCopyMultiple` (page 836)
Returns a dictionary containing preference values for multiple keys.

`CFPreferencesCopyValue` (page 837)
Returns a preference value for a given domain.

`CFPreferencesGetAppBooleanValue` (page 838)
Convenience function that directly obtains a boolean preference value for the specified key.

`CFPreferencesGetAppIntegerValue` (page 839)
Convenience function that directly obtains an integer preference value for the specified key.

Setting Preference Values

`CFPreferencesSetAppValue` (page 840)
Adds, modifies, or removes a preference.

`CFPreferencesSetMultiple` (page 841)
Convenience function that allows you to set and remove multiple preference values.

`CFPreferencesSetValue` (page 841)
Adds, modifies, or removes a preference value for the specified domain.

Synchronizing Preferences

`CFPreferencesAppSynchronize` (page 833)
Writes to permanent storage all pending changes to the preference data for the application, and reads the latest preference data from permanent storage.

`CFPreferencesSynchronize` (page 842)
For the specified domain, writes all pending changes to preference data to permanent storage, and reads latest preference data from permanent storage.

Adding and Removing Suite Preferences

`CFPreferencesAddSuitePreferencesToApp` (page 832)
Adds suite preferences to an application's preference search chain.

`CFPreferencesRemoveSuitePreferencesFromApp` (page 839)
Removes suite preferences from an application's search chain.

Miscellaneous Functions

`CFPreferencesAppValueIsForced` (page 834)
Determines whether or not a given key has been imposed on the user.

`CFPreferencesCopyApplicationList` (page 834)
Constructs and returns the list of all applications that have preferences in the scope of the specified user and host.

Functions

CFPreferencesAddSuitePreferencesToApp

Adds suite preferences to an application's preference search chain.

```
void CFPreferencesAddSuitePreferencesToApp (
   CFStringRef applicationID,
   CFStringRef suiteID
);
```

Parameters

applicationID

The ID of the application to which to add suite preferences, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, `com.foosoft`.

suiteID

The ID of the application suite preferences to add. Takes the form of a Java package name, `com.foosoft`.

Discussion

Suite preferences allow you to maintain a set of preferences that are common to all applications in the suite. When a suite is added to an application's search chain, all of the domains pertaining to that suite are inserted into the chain. Suite preferences are added between the "Current Application" domains and the "Any Application" domains. If you add multiple suite preferences to one application, the order of the suites in the search chain is non-deterministic. You can override a suite preference for a given application by defining the same preference key in the application specific preferences.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFPreferences.h`

CFPreferencesAppSynchronize

Writes to permanent storage all pending changes to the preference data for the application, and reads the latest preference data from permanent storage.

```
Boolean CFPreferencesAppSynchronize (
   CFStringRef applicationID
);
```

Parameters

applicationID

The ID of the application whose preferences to write to storage, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, `com.foosoft`.

Return Value

`true` if synchronization was successful, otherwise `false`.

Discussion

Calling the function `CFPreferencesSetAppValue` (page 840) is not in itself sufficient for storing preferences. The `CFPreferencesAppSynchronize` function writes to permanent storage all pending preference changes for the application. Typically you would call this function after multiple calls to `CFPreferencesSetAppValue` (page 840). Conversely, preference data is cached after it is first read. Changes made externally are not automatically incorporated. The `CFPreferencesAppSynchronize` function reads the latest preferences from permanent storage.

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

Related Sample Code
ConvertMovieSndTrack
HID Config Save
HID Dumper
Preferences
RecentItems

Declared In
`CFPreferences.h`

CFPreferencesAppValueIsForced

Determines whether or not a given key has been imposed on the user.

```
Boolean CFPreferencesAppValueIsForced (
   CFStringRef key,
   CFStringRef applicationID
);
```

Parameters

key
The key you are querying.

applicationID
The application's ID, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, `com.foosoft`.

Return Value
`true` if value of the key cannot be changed by the user, otherwise `false`.

Discussion
In cases where machines and/or users are under some kind of management, you should use this function to determine whether or not to disable UI elements corresponding to those preference keys.

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

Declared In
`CFPreferences.h`

CFPreferencesCopyApplicationList

Constructs and returns the list of all applications that have preferences in the scope of the specified user and host.

```
CFArrayRef CFPreferencesCopyApplicationList (
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

userName

`kCFPreferencesCurrentUser` (page 844) to search the current-user domain, otherwise `kCFPreferencesAnyUser` (page 844) to search the any-user domain.

hostName

`kCFPreferencesCurrentHost` (page 844) to search the current-host domain, otherwise `kCFPreferencesAnyHost` (page 844) to search the any-host domain.

Return Value
The list of application IDs. Ownership follows the Create Rule.

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

Related Sample Code
CFPrefsDumper

Declared In
`CFPreferences.h`

CFPreferencesCopyAppValue

Obtains a preference value for the specified key and application.

```
CFPropertyListRef CFPreferencesCopyAppValue (
   CFStringRef key,
   CFStringRef applicationID
);
```

Parameters

key

The preference key whose value to obtain.

applicationID

The identifier of the application whose preferences to search, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, `com.foosoft`.

Return Value
The preference data for the specified key and application. If no value was located, returns `NULL`. Ownership follows the Create Rule.

Discussion
Note that values returned from this function are immutable, even if you have recently set the value using a mutable object.

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

Related Sample Code
CFPrefTopScores
ConvertMovieSndTrack
DockBrowser
Preferences
RecentItems

Declared In
`CFPreferences.h`

CFPreferencesCopyKeyList

Constructs and returns the list of all keys set in the specified domain.

```
CFArrayRef CFPreferencesCopyKeyList (
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

applicationID
The ID of the application whose preferences to search. Takes the form of a Java package name, `com.foosoft`.

userName
`kCFPreferencesCurrentUser` (page 844) to search the current-user domain, otherwise `kCFPreferencesAnyUser` (page 844) to search the any-user domain.

hostName
`kCFPreferencesCurrentHost` (page 844) to search the current-host domain, otherwise `kCFPreferencesAnyHost` (page 844) to search the any-host domain.

Return Value
The list of keys. Ownership follows the Create Rule.

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

Declared In
`CFPreferences.h`

CFPreferencesCopyMultiple

Returns a dictionary containing preference values for multiple keys.

```
CFDictionaryRef CFPreferencesCopyMultiple (
   CFArrayRef keysToFetch,
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

keysToFetch

An array of preference keys the values of which to obtain.

applicationID

The ID of the application whose preferences are searched. Takes the form of a Java package name, such as `com.foosoft`.

userName

`kCFPreferencesCurrentUser` (page 844) to search the current-user domain, otherwise `kCFPreferencesAnyUser` (page 844) to search the any-user domain.

hostName

`kCFPreferencesCurrentHost` (page 844) to search the current-host domain, otherwise `kCFPreferencesAnyHost` (page 844) to search the any-host domain.

Return Value

A dictionary containing the preference values for the keys specified by *keysToFetch* for the specified domain. If no values were located, returns an empty dictionary. Ownership follows the Create Rule.

Discussion

Note that values returned from this function are immutable, even if you have recently set the value using a mutable object.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPrefsDumper

Declared In

`CFPreferences.h`

CFPreferencesCopyValue

Returns a preference value for a given domain.

```
CFPropertyListRef CFPreferencesCopyValue (
   CFStringRef key,
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

key

Preferences key for the value to obtain.

applicationID
The ID of the application whose preferences are searched. Takes the form of a Java package name, such as `com.foosoft`.

userName
`kCFPreferencesCurrentUser` (page 844) if to search the current-user domain, otherwise `kCFPreferencesAnyUser` (page 844) to search the any-user domain.

hostName
`kCFPreferencesCurrentHost` (page 844) if to search the current-host domain, otherwise `kCFPreferencesAnyHost` (page 844) to search the any-host domain.

Return Value
The preference data for the specified domain. If the no value was located, returns `NULL`. Ownership follows the Create Rule.

Discussion
This function is the primitive get mechanism for the higher level preference function `CFPreferencesCopyAppValue` (page 835) Unlike the high-level function, `CFPreferencesCopyValue` (page 837) searches only the exact domain specified. Do not use this function directly unless you have a need. All arguments must be non-`NULL`. Do not use arbitrary user and host names, instead pass the pre-defined domain qualifier constants.

Note that values returned from this function are immutable, even if you have recently set the value using a mutable object.

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

Declared In
`CFPreferences.h`

CFPreferencesGetAppBooleanValue

Convenience function that directly obtains a boolean preference value for the specified key.

```
Boolean CFPreferencesGetAppBooleanValue (
   CFStringRef key,
   CFStringRef applicationID,
   Boolean *keyExistsAndHasValidFormat
);
```

Parameters

key
The preference key whose value to obtain. The key must specify a preference whose value is of type `Boolean`.

applicationID
The identifier of the application whose preferences are searched, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, such as `com.foosoft`.

keyExistsAndHasValidFormat
On return, `true` if the preference value for the specified key was located and found to be of type `Boolean`, otherwise `false`.

Return Value
The preference data for the specified key and application, or if no value was located, `false`.

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

Related Sample Code
RecentItems

Declared In
`CFPreferences.h`

CFPreferencesGetAppIntegerValue

Convenience function that directly obtains an integer preference value for the specified key.

```
CFIndex CFPreferencesGetAppIntegerValue (
   CFStringRef key,
   CFStringRef applicationID,
   Boolean *keyExistsAndHasValidFormat
);
```

Parameters

key
The preference key whose value you wish to obtain. The key must specify a preference whose value is of type `int`.

applicationID
The identifier of the application whose preferences you wish to search, typically `kCFPreferencesCurrentApplication` (page 844). Do not pass `NULL` or `kCFPreferencesAnyApplication` (page 844). Takes the form of a Java package name, `com.foosoft`.

keyExistsAndHasValidFormat
On return, indicates whether the preference value for the specified key was located and found to be of type `int`.

Return Value
The preference data for the specified key and application. If no value was located, `0` is returned.

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

Declared In
`CFPreferences.h`

CFPreferencesRemoveSuitePreferencesFromApp

Removes suite preferences from an application's search chain.

```
void CFPreferencesRemoveSuitePreferencesFromApp (
   CFStringRef applicationID,
   CFStringRef suiteID
);
```

Parameters

applicationID

The ID of the application from which to remove suite preferences, typically kCFPreferencesCurrentApplication (page 844). Do not pass NULL or kCFPreferencesAnyApplication (page 844). Takes the form of a Java package name, com.foosoft.

suiteID

The ID of the application suite preferences to remove. Takes the form of a Java package name, com.foosoft.

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

Declared In
CFPreferences.h

CFPreferencesSetAppValue

Adds, modifies, or removes a preference.

```
void CFPreferencesSetAppValue (
   CFStringRef key,
   CFPropertyListRef value,
   CFStringRef applicationID
);
```

Parameters

key

The preference key whose value you wish to set.

value

The value to set for the specified *key* and application. Pass NULL to remove the specified key from the application's preferences.

applicationID

The ID of the application whose preferences you wish to create or modify, typically kCFPreferencesCurrentApplication (page 844). Do not pass NULL or kCFPreferencesAnyApplication (page 844). Takes the form of a Java package name, com.foosoft.

Discussion
New preference values are stored in the standard application preference location, ~/Library/Preferences/. When called with kCFPreferencesCurrentApplication (page 844), modifications are performed in the preference domain "Current User, Current Application, Any Host." If you need to create preferences in some other domain, use the low-level function CFPreferencesSetValue (page 841).

You must call the CFPreferencesAppSynchronize (page 833) function in order for your changes to be saved to permanent storage.

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

Related Sample Code
DockBrowser
HID Config Save
HID Dumper
HID Explorer
RecentItems

Declared In
`CFPreferences.h`

CFPreferencesSetMultiple

Convenience function that allows you to set and remove multiple preference values.

```
void CFPreferencesSetMultiple (
   CFDictionaryRef keysToSet,
   CFArrayRef keysToRemove,
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

keysToSet
A dictionary containing the key/value pairs for the preferences to set.

keysToRemove
An array containing a list of keys to remove.

applicationID
The ID of the application whose preferences you wish to modify. Takes the form of a Java package name, `com.foosoft`.

userName
`kCFPreferencesCurrentUser` (page 844) to modify the current user's preferences, otherwise `kCFPreferencesAnyUser` (page 844) to modify the preferences of all users.

hostName
`kCFPreferencesCurrentHost` (page 844) to modify the preferences of the current host, otherwise `kCFPreferencesAnyHost` (page 844) to modify the preferences of all hosts.

Discussion
Behavior is undefined if a key is in both *keysToSet* and *keysToRemove*

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

Declared In
`CFPreferences.h`

CFPreferencesSetValue

Adds, modifies, or removes a preference value for the specified domain.

```
void CFPreferencesSetValue (
   CFStringRef key,
   CFPropertyListRef value,
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

key

Preferences key for the value you wish to set.

value

The value to set for *key* and application. Pass `NULL` to remove *key* from the domain.

applicationID

The ID of the application whose preferences you wish to modify. Takes the form of a Java package name, `com.foosoft`.

userName

`kCFPreferencesCurrentUser` (page 844) to modify the current user's preferences, otherwise `kCFPreferencesAnyUser` (page 844) to modify the preferences of all users.

hostName

`kCFPreferencesCurrentHost` (page 844) to modify the preferences of the current host, otherwise `kCFPreferencesAnyHost` (page 844) to modify the preferences of all hosts.

Discussion

This function is the primitive set mechanism for the higher level preference function `CFPreferencesSetAppValue` (page 840). Only the exact domain specified is modified. Do not use this function directly unless you have a specific need. All arguments except `value` must be non-`NULL`. Do not use arbitrary user and host names, instead pass the pre-defined constants.

You must call the `CFPreferencesSynchronize` (page 842) function in order for your changes to be saved to permanent storage. Note that you can only save preferences for "Any User" if you have Admin privileges.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPrefTopScores

SampleRaster

Declared In

`CFPreferences.h`

CFPreferencesSynchronize

For the specified domain, writes all pending changes to preference data to permanent storage, and reads latest preference data from permanent storage.

```
Boolean CFPreferencesSynchronize (
   CFStringRef applicationID,
   CFStringRef userName,
   CFStringRef hostName
);
```

Parameters

applicationID

The ID of the application whose preferences you wish to modify. Takes the form of a Java package name, `com.foosoft`.

userName

`kCFPreferencesCurrentUser` (page 844) to modify the current user's preferences, otherwise `kCFPreferencesAnyUser` (page 844) to modify the preferences of all users.

hostName

`kCFPreferencesCurrentHost` (page 844) to search the current-host domain, otherwise `kCFPreferencesAnyHost` (page 844) to search the any-host domain.

Return Value

`true` if synchronization was successful, `false` if an error occurred.

Discussion

This function is the primitive synchronize mechanism for the higher level preference function `CFPreferencesAppSynchronize` (page 833); it writes updated preferences to permanent storage, and reads the latest preferences from permanent storage. Only the exact domain specified is modified. Note that to modify "Any User" preferences requires Admin privileges—see *Authorization Services Programming Guide*.

Do not use this function directly unless you have a specific need. All arguments must be non- `NULL`. Do not use arbitrary user and host names, instead pass the pre-defined constants.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPrefTopScores

Declared In

`CFPreferences.h`

Constants

Application, Host, and User Keys

Keys used to specify the common preference domains.

```
const CFStringRef kCFPreferencesAnyApplication;
const CFStringRef kCFPreferencesAnyHost;
const CFStringRef kCFPreferencesAnyUser;
const CFStringRef kCFPreferencesCurrentApplication;
const CFStringRef kCFPreferencesCurrentHost;
const CFStringRef kCFPreferencesCurrentUser;
```

Constants

`kCFPreferencesAnyApplication`
Indicates a preference that applies to any application.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

`kCFPreferencesAnyHost`
Indicates a preference that applies to any host.
This domain is currently unsupported.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

`kCFPreferencesAnyUser`
Indicates a preference that applies to any user.
This domain is currently unsupported.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

`kCFPreferencesCurrentApplication`
Indicates a preference that applies only to the current application.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

`kCFPreferencesCurrentHost`
Indicates a preference that applies only to the current host.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

`kCFPreferencesCurrentUser`
Indicates a preference that applies only to the current user.
Available in Mac OS X v10.0 and later.
Declared in `CFPreferences.h`.

Socket Name Server Utilities Reference

Framework: CoreFoundation/CoreFoundation.h

Declared in CFSocket.h

Overview

Name server functionality is currently inoperable in Mac OS X.

Functions

CFSocketCopyRegisteredSocketSignature

Returns a socket signature registered with a CFSocket name server.

```
CFSocketError CFSocketCopyRegisteredSocketSignature (
   const CFSocketSignature *nameServerSignature,
   CFTimeInterval timeout,
   CFStringRef name,
   CFSocketSignature *signature,
   CFDataRef *nameServerAddress
);
```

Parameters

nameServerSignature

The socket signature for the name server. If `NULL`, this function contacts the default server, which is assumed to be a local process using TCP/IP to listen on the port number returned from `CFSocketGetDefaultNameRegistryPortNumber` (page 847). If *nameServerSignature* is incomplete, the missing values are replaced with the default server's values, if appropriate.

timeout

The time to wait for the server to accept a connection and to reply to the registration request.

name

The name of the registered socket signature to retrieve.

signature

A pointer to a `CFSocketSignature` structure into which the retrieved socket signature is copied.

nameServerAddress

A pointer to a CFData object into which the name server's address is copied. Pass `NULL` if you do not want the server's address.

Return Value
An error code indicating success or failure.

Discussion
Once you have the socket signature, you can open a connection to that socket with `CFSocketCreateConnectedToSocketSignature` (page 519).

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

Declared In
`CFSocket.h`

CFSocketCopyRegisteredValue

Returns a value registered with a CFSocket name server.

```
CFSocketError CFSocketCopyRegisteredValue (
   const CFSocketSignature *nameServerSignature,
   CFTimeInterval timeout,
   CFStringRef name,
   CFPropertyListRef *value,
   CFDataRef *nameServerAddress
);
```

Parameters

nameServerSignature
The socket signature for the name server. If `NULL`, this function contacts the default server, which is assumed to be a local process using TCP/IP to listen on the port number returned from `CFSocketGetDefaultNameRegistryPortNumber` (page 847). If *nameServerSignature* is incomplete, the missing values are replaced with the default server's values, if appropriate.

timeout
The time to wait for the server to accept a connection and to reply to the registration request.

name
The name of the registered value to return.

value
A pointer to the property list object into which the retrieved value should be copied.

nameServerAddress
A pointer to a CFData object into which the name server's address is copied. Pass `NULL` if you do not want the server's address.

Return Value
An error code indicating success or failure.

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

Declared In
`CFSocket.h`

CFSocketGetDefaultNameRegistryPortNumber

Returns the default port number with which to connect to a CFSocket name server.

```
UInt16 CFSocketGetDefaultNameRegistryPortNumber (
   void
);
```

Return Value
The default port number with which to connect to a CFSocket name server.

Discussion
If you do not provide a name server signature or leave out the socket address in the signature when calling one of the name registry functions, such as `CFSocketRegisterSocketSignature` (page 847), the returned port number is used for the connection.

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

Declared In
`CFSocket.h`

CFSocketRegisterSocketSignature

Registers a socket signature with a CFSocket name server.

```
CFSocketError CFSocketRegisterSocketSignature (
   const CFSocketSignature *nameServerSignature,
   CFTimeInterval timeout,
   CFStringRef name,
   const CFSocketSignature *signature
);
```

Parameters

nameServerSignature
The socket signature for the name server. If `NULL`, this function contacts the default server, which is assumed to be a local process using TCP/IP to listen on the port number returned from `CFSocketGetDefaultNameRegistryPortNumber` (page 847). If *nameServerSignature* is incomplete, the missing values are replaced with the default server's values, if appropriate.

timeout
The time to wait for the server to accept a connection and to reply to the registration request.

name
The name with which to register *signature*.

signature
The socket signature to register.

Return Value
An error code indicating success or failure.

Discussion
Once a socket signature is registered, other processes can retrieve it with `CFSocketCopyRegisteredSocketSignature` (page 845) and then open a connection to your socket using `CFSocketCreateConnectedToSocketSignature` (page 519).

To remove a registered socket signature from the name server, use `CFSocketUnregister` (page 849).

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

Declared In
`CFSocket.h`

CFSocketRegisterValue

Registers a property-list value with a CFSocket name server.

```
CFSocketError CFSocketRegisterValue (
   const CFSocketSignature *nameServerSignature,
   CFTimeInterval timeout,
   CFStringRef name,
   CFPropertyListRef value
);
```

Parameters

nameServerSignature
The socket signature for the name server. If `NULL`, this function contacts the default server, which is assumed to be a local process using TCP/IP to listen on the port number returned from `CFSocketGetDefaultNameRegistryPortNumber` (page 847). If *nameServerSignature* is incomplete, the missing values are replaced with the default server's values, if appropriate.

timeout
The time to wait for the server to accept a connection and to reply to the registration request.

name
The name with which to register *value*.

value
The property-list value to register.

Return Value
An error code indicating success or failure.

Discussion
To remove a registered value from the name server, use `CFSocketUnregister` (page 849).

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

Declared In
`CFSocket.h`

CFSocketSetDefaultNameRegistryPortNumber

Sets the default port number with which to connect to a CFSocket name server.

```
void CFSocketSetDefaultNameRegistryPortNumber (
   UInt16 port
);
```

Parameters

port

The port number to use to connect to the CFSocket name server.

Discussion

If you do not provide a name server signature or leave out the socket address in the signature when calling one of the name registry functions, such as `CFSocketRegisterSocketSignature` (page 847), *port* will be used for the connection.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketUnregister

Unregisters a value or socket signature with a CFSocket name server.

```
CFSocketError CFSocketUnregister (
   const CFSocketSignature *nameServerSignature,
   CFTimeInterval timeout,
   CFStringRef name
);
```

Parameters

nameServerSignature

The socket signature for the name server. If `NULL`, this function contacts the default server, which is assumed to be a local process using TCP/IP to listen on the port number returned from `CFSocketGetDefaultNameRegistryPortNumber` (page 847). If *nameServerSignature* is incomplete, the missing values are replaced with the default server's values, if appropriate.

timeout

The time to wait for the server to accept a connection and to reply to the registration request.

name

The name of the property-list value or socket signature to unregister.

Return Value

An error code indicating success or failure.

Discussion

The value being unregistered was previously registered with `CFSocketRegisterValue` (page 848) or `CFSocketRegisterSocketSignature` (page 847).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

Constants

CFSocket Name Server Keys

Not used.

```
const CFStringRef kCFSocketCommandKey;
const CFStringRef kCFSocketNameKey;
const CFStringRef kCFSocketValueKey;
const CFStringRef kCFSocketResultKey;
const CFStringRef kCFSocketErrorKey;
const CFStringRef kCFSocketRegisterCommand;
const CFStringRef kCFSocketRetrieveCommand;
```

Constants

`kCFSocketCommandKey`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketNameKey`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketValueKey`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketResultKey`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketErrorKey`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketRegisterCommand`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

`kCFSocketRetrieveCommand`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CFSocket.h`.

Declared In

`CFSocket.h`

Time Utilities Reference

Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFDate.h
Companion guide	Date and Time Programming Guide for Core Foundation

Overview

Core Foundation measures time in units of seconds. The base data type is the `CFTimeInterval` (page 858), which measures the difference in seconds between two times. Fixed times, or dates, are defined by the `CFAbsoluteTime` (page 856) data type, which measures the time interval between a particular date and the absolute reference date of Jan 1 2001 00:00:00 GMT.

The `CFGregorianDate` (page 857) structure represents absolute times in terms of the Gregorian calendar. Functions such as `CFAbsoluteTimeGetGregorianDate` (page 854) use a CFTimeZone object to obtain the local time in a particular time zone.

The CFDate opaque type wraps an absolute time into a CFType-base object, allowing you to put time objects into into collections and property lists and to be handled by other object-oriented parts of Core Foundation.

Functions

CFAbsoluteTimeAddGregorianUnits

Adds a time interval, expressed as Gregorian units, to a given absolute time.

```
CFAbsoluteTime CFAbsoluteTimeAddGregorianUnits (
    CFAbsoluteTime at,
    CFTimeZoneRef tz,
    CFGregorianUnits units
);
```

Parameters

at

The absolute time to which the interval is added.

tz

The time zone to use for time correction. Pass `NULL` for GMT.

units

The time interval to add to *at*.

Return Value
An absolute time value equal to the sum of *at* and *units*.

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

Declared In
CFDate.h

CFAbsoluteTimeGetCurrent

Returns the current system absolute time.

```
CFAbsoluteTime CFAbsoluteTimeGetCurrent ();
```

Return Value
The current absolute time.

Discussion
Absolute time is measured in seconds relative to the absolute reference date of Jan 1 2001 00:00:00 GMT. A positive value represents a date after the reference date, a negative value represents a date before it. For example, the absolute time -32940326 is equivalent to December 16th, 1999 at 17:54:34. Repeated calls to this function do not guarantee monotonically increasing results. The system time may decrease due to synchronization with external time references or due to an explicit user change of the clock.

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

Related Sample Code
AudioQueueTools
Cocoa OpenGL
ColorSyncDevices
MFSLives
Worm

Declared In
CFDate.h

CFAbsoluteTimeGetDayOfWeek

Returns an integer representing the day of the week indicated by the specified absolute time.

```
SInt32 CFAbsoluteTimeGetDayOfWeek (
    CFAbsoluteTime at,
    CFTimeZoneRef tz
);
```

Parameters

at
The absolute time to convert.

tz
The time zone to use for time correction. Pass NULL for GMT.

Return Value
An integer (`1-7`) representing the day of the week specified by *at*. Per ISO-8601, Monday is represented by `1`, Tuesday by `2`, and so on.

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

Declared In
`CFDate.h`

CFAbsoluteTimeGetDayOfYear

Returns an integer representing the day of the year indicated by the specified absolute time.

```
SInt32 CFAbsoluteTimeGetDayOfYear (
    CFAbsoluteTime at,
    CFTimeZoneRef tz
);
```

Parameters

at
The absolute time to convert.

tz
The time zone to use for time correction. Pass `NULL` for GMT.

Return Value
An integer (`1-366`) representing the day of the year specified by *at*.

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

Declared In
`CFDate.h`

CFAbsoluteTimeGetDifferenceAsGregorianUnits

Computes the time difference between two specified absolute times and returns the result as an interval in Gregorian units.

```
CFGregorianUnits CFAbsoluteTimeGetDifferenceAsGregorianUnits (
    CFAbsoluteTime at1,
    CFAbsoluteTime at2,
    CFTimeZoneRef tz,
    CFOptionFlags unitFlags
);
```

Parameters

at1
An absolute time.

at2
An absolute time.

tz

The time zone to use for time correction. Pass `NULL` for GMT.

unitFlags

A mask that specifies which Gregorian unit fields to use when converting the absolute time difference into a Gregorian interval. See Gregorian Unit Flags (page 858) for a list of values from which to construct the mask.

Return Value
The difference between the specified absolute times (as `at1 - at2`—if *at1* is earlier than *at2*, the result is negative) expressed in the units specified by *unitFlags*.

Discussion
The temporal difference is expressed as accurately as possible, given the units specified. For example, if you asked for the number of months and hours between 2:30pm on April 8 2005 and 5:45pm September 9 2005, the result would be 5 months and 27 hours.

The following example prints the number of hours and minutes between the current time (now) and the reference date (1 January 2001 00:00:00 GMT).

```
CFAbsoluteTime now = CFAbsoluteTimeGetCurrent ();

CFGregorianUnits units = CFAbsoluteTimeGetDifferenceAsGregorianUnits
    (now, 0, NULL, (kCFGregorianUnitsHours | kCFGregorianUnitsMinutes));

CFStringRef output = CFStringCreateWithFormat
    (NULL, 0, CFSTR("hours: %d; minutes: %d"), units.hours, units.minutes);
CFShow(output);
```

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

Declared In
`CFDate.h`

CFAbsoluteTimeGetGregorianDate

Converts an absolute time value into a Gregorian date.

```
CFGregorianDate CFAbsoluteTimeGetGregorianDate (
    CFAbsoluteTime at,
    CFTimeZoneRef tz
);
```

Parameters

at

The absolute time value to convert.

tz

The time zone to use for time correction. Pass `NULL` for GMT.

Return Value
The Gregorian date equivalent for *at*.

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

Related Sample Code
SampleDS

Declared In
`CFDate.h`

CFAbsoluteTimeGetWeekOfYear

Returns an integer representing the week of the year indicated by the specified absolute time.

```
SInt32 CFAbsoluteTimeGetWeekOfYear (
    CFAbsoluteTime at,
    CFTimeZoneRef tz
);
```

Parameters

at
The absolute time to convert.

tz
The time zone to use for time correction. Pass `NULL` for GMT.

Return Value
An integer (`1-53`) representing the week of the year specified by *at*. The numbering follows the ISO 8601 definition of week.

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

Declared In
`CFDate.h`

CFGregorianDateGetAbsoluteTime

Converts a Gregorian date value into an absolute time value.

```
CFAbsoluteTime CFGregorianDateGetAbsoluteTime (
    CFGregorianDate gdate,
    CFTimeZoneRef tz
);
```

Parameters

gdate
The Gregorian date to convert.

tz
The time zone to use for time correction. Pass `NULL` for GMT.

Return Value
The absolute time equivalent of *gdate*.

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

Declared In
`CFDate.h`

CFGregorianDateIsValid

Checks the specified fields of a CFGregorianDate structure for valid values.

```
Boolean CFGregorianDateIsValid (
    CFGregorianDate gdate,
    CFOptionFlags unitFlags
);
```

Parameters

gdate

The CFGregorianDate structure whose fields to validate.

unitFlags

A mask that specifies which Gregorian unit fields to validate. See Gregorian Unit Flags (page 858) for a list of values from which to construct the mask.

Return Value
`true` if the specified fields are valid, otherwise `false`.

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

Declared In
`CFDate.h`

Data Types

CFAbsoluteTime

Type used to represent a specific point in time relative to the absolute reference date of 1 Jan 2001 00:00:00 GMT.

```
typedef CFTimeInterval CFAbsoluteTime;
```

Discussion
Absolute time is measured by the number of seconds between the reference date and the specified date. Negative values indicate dates/times before the reference date. Positive values indicate dates/times after the reference date.

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

Declared In
`CFDate.h`

CFGregorianDate

Structure used to represent a point in time using the Gregorian calendar.

```
struct CFGregorianDate {
    SInt32 year;
    SInt8 month;
    SInt8 day;
    SInt8 hour;
    SInt8 minute;
    double second;
};
typedef struct CFGregorianDate CFGregorianDate;
```

Discussion
CFGregorianDate is implemented using the smallest data type appropriate for the range of possible values. For example, there are only 12 months in the Gregorian year, so there is no need to use an integer type larger than 8 bits. To represent a time interval in Gregorian units, use a `CFGregorianUnits` (page 857).

The month and day units are 1-based: the index for January is 1, and the index for the first day of the month is 1.

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

Declared In
`CFDate.h`

CFGregorianUnits

Structure used to represent a time interval in Gregorian units.

```
struct CFGregorianUnits {
    SInt32 years;
    SInt32 months;
    SInt32 days;
    SInt32 hours;
    SInt32 minutes;
    double seconds;
};
typedef struct CFGregorianUnits CFGregorianUnits;
```

Discussion
A CFGregorianUnits is used to represent arbitrary time *intervals* (to represent a point in time using Gregorian units, use a `CFGregorianDate` (page 857)). Each field can take values up to the maximum possible for its data type. Negative values are also valid.

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

Declared In
`CFDate.h`

CFTimeInterval

Type used to represent elapsed time in seconds.

```
typedef double CFTimeInterval;
```

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

Declared In
`CFDate.h`

Constants

CFGregorianUnitFlags

These option flags are used as a mask to indicate a specific set of fields in the CFGregorianDate or CFGregorianUnits structures.

```
enum CFGregorianUnitFlags {
    kCFGregorianUnitsYears = (1 << 0),
    kCFGregorianUnitsMonths = (1 << 1),
    kCFGregorianUnitsDays = (1 << 2),
    kCFGregorianUnitsHours = (1 << 3),
    kCFGregorianUnitsMinutes = (1 << 4),
    kCFGregorianUnitsSeconds = (1 << 5),
    kCFGregorianAllUnits = 0x00FFFFFF
};
typedef enum CFGregorianUnitFlags CFGregorianUnitFlags;
```

Constants

`kCFGregorianUnitsYears`
Specifies the year field.
Available in Mac OS X v10.0 and later.
Declared in `CFDate.h`.

`kCFGregorianUnitsMonths`
Specifies the month field.
Available in Mac OS X v10.0 and later.
Declared in `CFDate.h`.

`kCFGregorianUnitsDays`
Specifies the day field.
Available in Mac OS X v10.0 and later.
Declared in `CFDate.h`.

`kCFGregorianUnitsHours`
Specifies the hours field.
Available in Mac OS X v10.0 and later.
Declared in `CFDate.h`.

`kCFGregorianUnitsMinutes`
Specifies the minutes field.

Available in Mac OS X v10.0 and later.

Declared in `CFDate.h`.

`kCFGregorianUnitsSeconds`
Specifies the seconds field.

Available in Mac OS X v10.0 and later.

Declared in `CFDate.h`.

`kCFGregorianAllUnits`
Specifies all fields.

Available in Mac OS X v10.0 and later.

Declared in `CFDate.h`.

Discussion
These flags are used with functions such as `CFGregorianDateIsValid` (page 856) and `CFAbsoluteTimeGetDifferenceAsGregorianUnits` (page 853) which operate on a CFGregorianDate or CFGregorianUnits structure. For more details, see the discussion of those functions.

Declared In
`CFDate.h`

Predefined Time Interval Values

Time intervals between the absolute reference date and certain other dates.

```
const CFTimeInterval kCFAbsoluteTimeIntervalSince1970;
const CFTimeInterval kCFAbsoluteTimeIntervalSince1904;
```

Constants
`kCFAbsoluteTimeIntervalSince1970`
The time interval between 1 January 1970 and the reference date 1 January 2001 00:00:00 GMT.

Available in Mac OS X v10.0 and later.

Declared in `CFDate.h`.

`kCFAbsoluteTimeIntervalSince1904`
The time interval between 1 January 1904 and the reference date 1 January 2001 00:00:00 GMT.

Available in Mac OS X v10.0 and later.

Declared in `CFDate.h`.

Other References

CFStream Reference

Framework:	CoreFoundation/CoreFoundation.h
Declared in	CoreFoundation/CFStream.h
Companion guides	Getting Started with Networking CFNetwork Programming Guide

Overview

This document describes the generic `CFStream` functions, data types, and constants. See also `CFReadStreamRef` (page 451) and `CFWriteStreamRef` (page 753) for functions and constants specific to read and write streams respectively.

Functions

CFStreamCreateBoundPair

Creates a pair of read and write streams.

```
void CFStreamCreateBoundPair (
   CFAllocatorRef alloc,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream,
   CFIndex transferBufferSize
);
```

Parameters

alloc

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

readStream

Upon return, a readable stream. Ownership follows the Create Rule.

writeStream

Upon return, a writable. Ownership follows the Create Rule.

transferBufferSize

The size of the buffer to use to transfer data from *readStream* to *writeStream*.

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

Declared In
CFStream.h

CFStreamCreatePairWithPeerSocketSignature

Creates readable and writable streams connected to a socket.

```
void CFStreamCreatePairWithPeerSocketSignature (
   CFAllocatorRef alloc,
   const CFSocketSignature *signature,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

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

signature
A CFSocketSignature structure identifying the communication protocol and address to which the socket streams should connect.

readStream
On return, a readable stream connected to the socket address in *signature*. If you pass NULL, this function will not create a readable stream. Ownership follows the Create Rule.

writeStream
On return, a writable stream connected to the socket address in *signature*. If you pass NULL, this function will not create a writable stream. Ownership follows the Create Rule.

Discussion
The streams do not open a connection to the socket until one of the streams is opened.

Most properties are shared by both streams. Setting the property for one stream automatically sets the property for the other.

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

Declared In
CFStream.h

CFStreamCreatePairWithSocket

Creates readable and writable streams connected to a socket.

```
void CFStreamCreatePairWithSocket (
    CFAllocatorRef alloc,
    CFSocketNativeHandle sock,
    CFReadStreamRef *readStream,
    CFWriteStreamRef *writeStream
);
```

Parameters

alloc

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

sock

The pre-existing (and already connected) socket which the socket streams should use.

readStream

Upon return, a readable stream connected to the socket address in *signature*. If you pass `NULL`, this function will not create a readable stream. Ownership follows the Create Rule.

writeStream

Upon return, a writable stream connected to the socket address in *signature*. If you pass `NULL`, this function will not create a writable stream. Ownership follows the Create Rule.

Discussion

Most properties are shared by both streams. Setting the property for one stream automatically sets the property for the other.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

CocoaEcho

CocoaHTTPServer

CocoaSOAP

Declared In

`CFStream.h`

CFStreamCreatePairWithSocketToHost

Creates readable and writable streams connected to a TCP/IP port of a particular host.

```
void CFStreamCreatePairWithSocketToHost (
   CFAllocatorRef alloc,
   CFStringRef host,
   UInt32 port,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

alloc

The allocator to use to allocate memory for the `CFReadStream` and `CFWriteStream` objects. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

host
The host name to which the socket streams should connect. The host can be specified using an IPv4 or IPv6 address or a fully qualified DNS host name.

port
The TCP port number to which the socket streams should connect.

readStream
Upon return, a readable stream connected to the socket address in *port*. If you pass NULL, this function will not create a readable stream. Ownership follows the Create Rule.

writeStream
Upon return, a writable stream connected to the socket address in *port*. If you pass NULL, this function will not create a writable stream. Ownership follows the Create Rule.

Discussion
The streams do not open a connection to the specified host until one of the streams is opened.

Most properties are shared by both streams. Setting the property for one stream automatically sets the property for the other.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CFStream.h

Data Types

CFStreamError

The structure returned by CFReadStreamGetError (page 444) and CFWriteStreamGetError (page 746). (Deprecated. Use CFReadStreamCopyError (page 441) and CFWriteStreamCopyError (page 743) instead.)

```
typedef struct {
CFStreamErrorDomain domain;
SInt32 error
} CFStreamError;
```

Fields
domain
The error domain that should be used to interpret the error. See CFStream Error Domain Constants (page 869) for possible values.

error
The error code.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CFStream.h

CFStreamClientContext

A structure provided when an application registers itself to receive stream-related events.

```
struct CFStreamClientContext {
    CFIndex version;
    void *info;
    void *(*retain)(void *info);
    void (*release)(void *info);
    CFStringRef (*copyDescription)(void *info);
} CFStreamClientContext;
```

Fields

`version`

An integer of type `CFIndex`. Currently the only valid value is zero.

`info`

A pointer to allocated memory containing user-defined data that will be valid for as long as the client is registered with the stream. You may assign `NULL` if your callback function doesn't want to receive user-defined data.

`retain`

A pointer to a function callback that retains the data pointed to by the `info` field.You may set this function pointer to `NULL`.

`release`

A pointer to a function callback that releases the data pointed to by the `info` field.You may set this function pointer to `NULL` but doing so might result in memory leaks.

`copyDescription`

A pointer to a function callback that provides a description of the data pointed to by the `info` field. In implementing this function, return a reference to a `CFString` object that describes your allocator, particularly some characteristics of your user-defined data. You may set this function pointer to `NULL`, in which case Core Foundation will provide a rudimentary description.

Declared In

`CoreFoundation/CFStream.h`

Constants

CFStream Status Constants

Constants that describe the status of a stream.

```
typedef enum {
    kCFStreamStatusNotOpen = 0,
    kCFStreamStatusOpening,
    kCFStreamStatusOpen,
    kCFStreamStatusReading,
    kCFStreamStatusWriting,
    kCFStreamStatusAtEnd,
    kCFStreamStatusClosed,
    kCFStreamStatusError
} CFStreamStatus;
```

Constants

`kCFStreamStatusNotOpen`
The stream is not open for reading or writing.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusOpening`
The stream is being opened for reading or for writing.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusOpen`
The stream is open.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusReading`
The stream is being read from.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusWriting`
The stream is being written to.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusAtEnd`
There is no more data to read, or no more data can be written.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusClosed`
The stream is closed.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamStatusError`
An error occurred on the stream.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

Discussion
The `CFStreamStatus` enumeration defines constants that describe the status of a stream. These values are returned by `CFReadStreamGetStatus` (page 445) and `CFWriteStreamGetStatus` (page 747).

Declared In
`CoreFoundation/CFStream.h`

CFStream Error Domain Constants

Defines constants for values returned in the domain field of the `CFStreamError` structure. (Deprecated. These constants are returned by `CFReadStreamGetError` (page 444) and `CFWriteStreamGetError` (page 746); use `CFReadStreamCopyError` (page 441) and `CFWriteStreamCopyError` (page 743) instead.)

```
typedef enum {
    kCFStreamErrorDomainCustom = -1,
    kCFStreamErrorDomainPOSIX = 1,
    kCFStreamErrorDomainMacOSStatus,
} CFStreamErrorDomain;
```

Constants

`kCFStreamErrorDomainCustom`
The error code is a custom error code.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamErrorDomainPOSIX`
The error code is an error code defined in `errno.h`.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamErrorDomainMacOSStatus`
The error is an OSStatus value defined in `MacErrors.h`.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

Discussion
These constants indicate how the error code in the `error` field in the `CFStreamError` (page 866) structure should be interpreted.

Declared In
`CoreFoundation/CFStream.h`

CFStream Error Domain Constants (CFHost)

Defines constants for values returned in the domain field of the `CFStreamError` structure.

```
const SInt32 kCFStreamErrorDomainNetDB;
const SInt32 kCFStreamErrorDomainNetServices;
const SInt32 kCFstreamErrorDomainMach;
const SInt32 kCFStreamErrorDomainFTP;
const SInt32 kCFStreamErrorDomainHTTP;
const int kCFStreamErrorDomainSOCKS;
const SInt32 kCFStreamErrorDomainSystemConfiguration;
const int kCFStreamErrorDomainSSL;
```

Constants

`kCFStreamErrorDomainNetDB`
The error code is an error code defined in `netdb.h`.
Available in Mac OS X v10.3 and later.
Declared in `CFHost.h`.

`kCFStreamErrorDomainNetServices`
The error code is a `CFNetService` error code. For details, see the `CFNetService Error Constants` enumeration.
Available in Mac OS X v10.2 and later.
Declared in `CFNetServices.h`.

`kCFStreamErrorDomainMach`
The error code is a Mach error code defined in `mach/error.h`.
Available in Mac OS X v10.2 and later.
Declared in `CFNetServices.h`.

`kCFStreamErrorDomainFTP`
The error code is an FTP error code.
Available in Mac OS X v10.3 and later.
Declared in `CFFTPStream.h`.

`kCFStreamErrorDomainHTTP`
The error code is an HTTP error code.
Available in Mac OS X v10.1 and later.
Declared in `CFHTTPStream.h`.

`kCFStreamErrorDomainSOCKS`
The error code is a SOCKS proxy error.
Available in Mac OS X v10.2 and later.
Declared in `CFSocketStream.h`.

`kCFStreamErrorDomainSystemConfiguration`
The error code is a system configuration error code as defined in `System/ConfigurationSystemConfiguration.h`.
Available in Mac OS X v10.3 and later.
Declared in `CFHost.h`.

`kCFStreamErrorDomainSSL`
The error code is an SSL error code as defined in `Security/SecureTransport.h`.
Available in Mac OS X v10.1 and later.
Declared in `CFSocketStream.h`.

Discussion
These constants indicate how the error code in the `error` field in the `CFStreamError` (page 866) structure should be interpreted.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CFNetwork.framework/CFHost.h`

CFStream Event Type Constants

Defines constants for stream-related events.

```
typedef enum {
kCFStreamEventNone = 0,
kCFStreamEventOpenCompleted = 1,
kCFStreamEventHasBytesAvailable = 2,
kCFStreamEventCanAcceptBytes = 4,
kCFStreamEventErrorOccurred = 8,
kCFStreamEventEndEncountered = 16
} CFStreamEventType;
```

Constants

`kCFStreamEventNone`
No event has occurred.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamEventOpenCompleted`
The open has completed successfully.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamEventHasBytesAvailable`
The stream has bytes to be read.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamEventCanAcceptBytes`
The stream can accept bytes for writing.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamEventErrorOccurred`
An error has occurred on the stream.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

`kCFStreamEventEndEncountered`
The end of the stream has been reached.
Available in Mac OS X v10.1 and later.
Declared in `CFStream.h`.

Discussion
This enumeration defines constants for stream-related events.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CoreFoundation/CFStream.h`

Stream Properties

Stream property names that can be set or copied.

```
const CFStringRef kCFStreamPropertyAppendToFile;
const CFStringRef kCFStreamPropertyFileCurrentOffset;
const CFStringRef kCFStreamPropertyDataWritten;
const CFStringRef kCFStreamPropertySocketNativeHandle;
const CFStringRef kCFStreamPropertySocketRemoteHostName;
const CFStringRef kCFStreamPropertySocketRemotePortNumber;
```

Constants

`kCFStreamPropertyDataWritten`
Value is a `CFData` object that contains all the bytes written to a writable memory stream. You cannot modify this value.

Available in Mac OS X v10.1 and later.

Declared in `CFStream.h`.

`kCFStreamPropertyAppendToFile`
Value is a `CFBoolean` value that indicates whether to append the written data to a file, if it already exists, rather than to replace its contents.

You must set this value before opening the writable file stream. The default value is `kCFBooleanFalse`, indicating that the stream should replace any pre-existing file. You cannot read this value.

Declared in `CFStream.h`.

Available in Mac OS X version 10.2 and later.

`kCFStreamPropertyFileCurrentOffset`
Value is a `CFNumber` object containing the current file offset.

Available in Mac OS X v10.3 and later.

Declared in `CFStream.h`.

`kCFStreamPropertySocketNativeHandle`
Value is a `CFData` object that contains the native handle for a socket stream—of type `CFSocketNativeHandle` (page 530)—to which the socket stream is connected.

This property is only available for socket streams. You cannot modify this value. You can read this value at any time.

Available in Mac OS X v10.1 and later.

Declared in `CFStream.h`.

`kCFStreamPropertySocketRemoteHostName`

Value is a `CFString` object containing the name of the host to which the socket stream is connected or `NULL` if unknown.

You cannot modify this value. You can read this value at any time.]

Available in Mac OS X v10.1 and later.

Declared in `CFStream.h`.

`kCFStreamPropertySocketRemotePortNumber`

Value is a `CFNumber` object containing the remote port number to which the socket stream is connected or `NULL` if unknown.

You cannot modify this value. You can read this value at any time.]

Available in Mac OS X v10.1 and later.

Declared in `CFStream.h`.

Discussion

Use `CFReadStreamCopyProperty` (page 442) or `CFWriteStreamCopyProperty` (page 744) to read the property values. Use `CFReadStreamSetProperty` (page 449) or `CFWriteStreamSetProperty` (page 750) to set the property values.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CoreFoundation/CFStream.h`

Document Revision History

This table describes the changes to *Core Foundation Framework Reference*.

Date	Notes
2007-10-31	Updated for Mac OS X v10.5. Added link to CFFileDescriptor reference.
2007-01-07	Includes new API for Leopard.
2006-05-23	First publication of this content as a collection of previously published documents.

Document Revision History