CFDictionary Reference

Core Foundation

2007-10-31

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

Simultaneously published in the United States and Canada.

Contents

CFDictionary Reference 5

Overview 5
Functions by Task 6
- Creating a dictionary 6
- Examining a dictionary 6
- Applying a function to a dictionary 6
- Getting the CFDictionary type ID 6

Functions 7
- CFDictionaryApplyFunction 7
- CFDictionaryContainsKey 7
- CFDictionaryContainsValue 8
- CFDictionaryCreate 9
- CFDictionaryCreateCopy 10
- CFDictionaryGetCount 11
- CFDictionaryGetCountOfKey 11
- CFDictionaryGetCountOfValue 12
- CFDictionaryGetKeysAndValues 12
- CFDictionaryGetTypeID 13
- CFDictionaryGetValue 14
- CFDictionaryGetValueIfPresent 14

Callbacks 15
- CFDictionaryApplierFunction 15
- CFDictionaryCopyDescriptionCallBack 16
- CFDictionaryEqualCallBack 17
- CFDictionaryHashCallBack 17
- CFDictionaryReleaseCallBack 18
- CFDictionaryRetainCallBack 18

Data Types 19
- CFDictionaryKeyCallBacks 19
- CFDictionaryRef 20
- CFDictionaryValueCallBacks 20

Constants 21
- Predefined Callback Structures 21

Document Revision History 23

Index 25

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 9) or `CFDictionaryCreateCopy` (page 10) function. Key-value pairs are passed as parameters to `CFDictionaryCreate` (page 9). 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 11) returns the number of key-value pairs in a dictionary; the `CFDictionaryContainsValue` (page 8) function checks if a value is in a dictionary; and `CFDictionaryGetKeysAndValues` (page 12) returns a C array containing all the values and a C array containing all the keys in a dictionary.

The `CFDictionaryApplyFunction` (page 7) 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 9)
Creates an immutable dictionary containing the specified key-value pairs.

`CFDictionaryCreateCopy` (page 10)
Creates and returns a new immutable dictionary with the key-value pairs of another dictionary.

Examining a dictionary

`CFDictionaryContainsKey` (page 7)
Returns a Boolean value that indicates whether a given key is in a dictionary.

`CFDictionaryContainsValue` (page 8)
Returns a Boolean value that indicates whether a given value is in a dictionary.

`CFDictionaryGetCount` (page 11)
Returns the number of key-value pairs in a dictionary.

`CFDictionaryGetCountOfKey` (page 11)
Returns the number of times a key occurs in a dictionary.

`CFDictionaryGetCountOfValue` (page 12)
Counts the number of times a given value occurs in the dictionary.

`CFDictionaryGetKeysAndValues` (page 12)
Fills two buffers with the keys and values from a dictionary.

`CFDictionaryGetValue` (page 14)
Returns the value associated with a given key.

`CFDictionaryGetValueIfPresent` (page 14)
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 7)
Calls a function once for each key-value pair in a dictionary.

Getting the CFDictionary type ID

`CFDictionaryGetTypeID` (page 13)
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

ColorSyncDevices

IOPrintSuperClasses

MoreIsBetter

MoreSCF

QISA

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

BSDLLCTest

MoreIsBetter

MoreSCF

QISA

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.

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 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 19) 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 19) 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 22) as this parameter to use the default callback functions.

valueCallBacks

A pointer to a `CFDictionaryValueCallBacks` (page 20) 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 22) 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

BSDLLCTest

HITextViewDemo

MoreIsBetter

MoreSCF

QISA

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` 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
MoreIsBetter
MoreSCF
QISA
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
MoreIsBetter
MoreSCF
QISA
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 Utilities Source
MoreIsBetter
MoreSCF
QISA

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 14) 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

databurntest

MoreIsBetter

MoreSCF

QISA

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
AlbumToSlideshow
HID Utilities Source
MoreIsBetter
MoreSCF
QISA

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 7) 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 9) in a `CFDictionaryKeyCallBacks` (page 19) structure or `CFDictionaryValueCallBacks` (page 20). This callback is used by the `CFCopyDescription` 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 9) in a `CFDictionaryKeyCallBacks` (page 19) and `CFDictionaryValueCallBacks` (page 20) 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 9) in a `CFDictionaryKeyCallBacks` (page 19) 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 9) in a `CFDictionaryKeyCallBacks` (page 19) and `CFDictionaryValueCallBacks` (page 20) 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 18) 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 18) 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 16) 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 17) 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 17) 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 18) 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 18) 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 16) 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 17) 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 19) 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` 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 19) 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 20) 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`.

Document Revision History

This table describes the changes to *CFDictionary Reference*.

Date	Notes
2007-10-31	Corrected minor typographical errors.
2007-01-08	Clarified comparison used by kCFCopyStringDictionaryKeyCallBacks.
2005-12-06	Made corrections in Companion Documents list.
2005-11-09	Corrected minor typographical errors.
2005-10-04	Corrected reversals of "key" and "value" in definitions of CFDictionaryKeyCallBacks and CFDictionaryValueCallBacks.
2005-08-11	Corrected minor typographical errors. Clarified use of strings for keys when generating XML.
2005-04-29	Moved Introduction to new Introduction page.
2004-10-05	Clarification of use of predefined callback structures.
2004-08-31	Correction to declaration of `CFDictionaryGetKeysAndValues` (page 12).
2004-04-22	Correction to declaration of return type of `CFDictionaryCopyDescriptionCallBack` example.
2003-08-01	Enhanced description of all the `kCFType*Callbacks` and added link to Carbon-Cocoa integration document and fixed errors.
2003-01-01	First version of this document.

Document Revision History

Index

C

CFDictionaryApplierFunction callback 15
CFDictionaryApplyFunction function 7
CFDictionaryContainsKey function 7
CFDictionaryContainsValue function 8
CFDictionaryCopyDescriptionCallBack callback 16
CFDictionaryCreate function 9
CFDictionaryCreateCopy function 10
CFDictionaryEqualCallBack callback 17
CFDictionaryGetCount function 11
CFDictionaryGetCountOfKey function 11
CFDictionaryGetCountOfValue function 12
CFDictionaryGetKeysAndValues function 12
CFDictionaryGetTypeID function 13
CFDictionaryGetValue function 14
CFDictionaryGetValueIfPresent function 14
CFDictionaryHashCallBack callback 17
CFDictionaryKeyCallBacks structure 19
CFDictionaryRef data type 20
CFDictionaryReleaseCallBack callback 18
CFDictionaryRetainCallBack callback 18
CFDictionaryValueCallBacks structure 20

K

kCFCopyStringDictionaryKeyCallBacks constant 22
kCFTypeDictionaryKeyCallBacks constant 22
kCFTypeDictionaryValueCallBacks constant 22

P

Predefined Callback Structures 21