CFBag Reference

Core Foundation

2007-05-22

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

Contents

CFBag Reference

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

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 7) or `CFBagCreateCopy` (page 8) 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 9) returns the number of values in a bag, the `CFBagContainsValue` (page 7) function checks if a value is in a bag, and `CFBagGetValues` (page 11) returns a C array containing all the values in a bag.

The `CFBagApplyFunction` (page 6) function lets you apply a function to all values in a bag.

Functions by Task

Creating a Bag

`CFBagCreate` (page 7)
Creates an immutable bag containing specified values.

`CFBagCreateCopy` (page 8)
Creates an immutable bag with the values of another bag.

Examining a Bag

`CFBagContainsValue` (page 7)
: Reports whether or not a value is in a bag.

`CFBagGetCount` (page 9)
: Returns the number of values currently in a bag.

`CFBagGetCountOfValue` (page 9)
: Returns the number of times a value occurs in a bag.

`CFBagGetValue` (page 10)
: Returns a requested value from a bag.

`CFBagGetValueIfPresent` (page 11)
: Reports whether or not a value is in a bag, and returns that value indirectly if it exists.

`CFBagGetValues` (page 11)
: Fills a buffer with values from a bag.

Applying a Function to the Contents of a Bag

`CFBagApplyFunction` (page 6)
: Calls a function once for each value in a bag.

Getting the CFBag Type ID

`CFBagGetTypeID` (page 10)
: 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 16) 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 16) 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 17) 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 6) 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 7) in a `CFBagCallBacks` (page 16) structure. This callback is used by the `CFCopyDescription` 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 7) in a `CFBagCallBacks` (page 16) 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 7) in a `CFBagCallBacks` (page 16) 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 7) in a `CFBagCallBacks` (page 16) 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 7) in a `CFBagCallBacks` (page 16) 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 15) 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 14) 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 12) 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 13) 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 14) 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 16) 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 16) 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`.

Document Revision History

This table describes the changes to *CFBag Reference*.

Date	Notes
2007-05-22	Made minor changes to text to conform to reference consistency guidelines.
2005-12-06	Made minor changes to text to conform to reference consistency guidelines.
2005-08-11	Cosmetic changes to conform to documentation guidelines.
2003-08-01	Enhanced description of all the `kCFType*Callbacks` and added link to Carbon-Cocoa integration document.
2003-01-01	First version of this document.

Index

C

K

P