CFSet Reference

Core Foundation

2005-12-06

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

Simultaneously published in the United States and Canada.

Contents

CFSet Reference

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

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 8) or `CFSetCreateCopy` (page 9) 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 9) returns the number of values in a set, the `CFSetContainsValue` (page 7) function checks if a value is in a set, and `CFSetGetValues` (page 12) 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 8)
Creates an immutable CFSet object containing supplied values.

`CFSetCreateCopy` (page 9)
Creates an immutable set containing the values of an existing set.

Examining a Set

`CFSetContainsValue` (page 7)
Returns a Boolean that indicates whether a set contains a given value.

`CFSetGetCount` (page 9)
Returns the number of values currently in a set.

`CFSetGetCountOfValue` (page 10)
Returns the number of values in a set that match a given value.

`CFSetGetValue` (page 11)
Obtains a specified value from a set.

`CFSetGetValueIfPresent` (page 11)
Reports whether or not a value is in a set, and if it exists returns the value indirectly.

`CFSetGetValues` (page 12)
Obtains all values in a set.

Applying a Function to Set Members

`CFSetApplyFunction` (page 6)
Calls a function once for each value in a set.

Getting the CFSet Type ID

`CFSetGetTypeID` (page 10)
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

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 16) 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 18) 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

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

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 6) 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 8) in a `CFSetCallBacks` (page 16) structure. This callback is used by the `CFCopyDescription` 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 8) in a `CFSetCallBacks` (page 16) 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 8) in a `CFSetCallBacks` (page 16) 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 8) in a `CFSetCallBacks` (page 16) 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 8) in a `CFSetCallBacks` (page 16) 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 16) 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 15) 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 13) 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 14) 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 14) 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 16) 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`, 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 `CFSet.h`.

`kCFCopyStringSetCallBacks`

Predefined `CFSetCallBacks` (page 16) 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`.

Document Revision History

This table describes the changes to *CFSet Reference*.

Date	Notes
2005-12-06	Made minor changes to text to conform to reference consistency guidelines.
	Made minor changes to text to conform to reference consistency guidelines.
2005-04-29	Moved Introduction to new Introduction page.
2004-08-31	Corrected declaration of `CFSetGetValues` (page 12).
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.

Document Revision History

Index

C

K

P