CFArray Reference

Core Foundation

2007-05-22

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

Simultaneously published in the United States and Canada.

Contents

CFArray Reference

Derived From:	CFPropertyList : CFType
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, 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 9) or `CFArrayCreateCopy` (page 10) 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` 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 11) and `CFArrayGetValueAtIndex` (page 14) provide the basis for all other functions in its interface. The `CFArrayGetCount` (page 11) function returns the number of elements in an array; `CFArrayGetValueAtIndex` (page 14) 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 7) lets you apply a function to values in an array, and `CFArrayBSearchValues` (page 7) 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 9)
Creates a new immutable array with the given values.

`CFArrayCreateCopy` (page 10)
Creates a new immutable array with the values from another array.

Examining an Array

`CFArrayBSearchValues` (page 7)
Searches an array for a value using a binary search algorithm.

`CFArrayContainsValue` (page 8)
Reports whether or not a value is in an array.

`CFArrayGetCount` (page 11)
Returns the number of values currently in an array.

`CFArrayGetCountOfValue` (page 12)
Counts the number of times a given value occurs in an array.

`CFArrayGetFirstIndexOfValue` (page 12)
Searches an array forward for a value.

`CFArrayGetLastIndexOfValue` (page 13)
Searches an array backward for a value.

`CFArrayGetValues` (page 15)
Fills a buffer with values from an array.

`CFArrayGetValueAtIndex` (page 14)
Retrieves a value at a given index.

Applying a Function to Elements

`CFArrayApplyFunction` (page 7)
Calls a function once for each element in range in an array.

Getting the CFArray Type ID

`CFArrayGetTypeID` (page 14)
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

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

HID Manager Basics

HID Utilities Source

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

MoreIsBetter

MoreSCF

QISA

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

MoreIsBetter

MoreSCF

PMPrinterPrintWithFile

QISA

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 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 19) 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 19) 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 kCFTypeArrayCallBacks (page 20) 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.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFPreferences

ImageClient

MoreIsBetter

MoreSCF

QISA

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` 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.

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
BSDLLCTest
ImageClient
MoreIsBetter
MoreSCF
QISA

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
MoreIsBetter
MoreSCF
QISA

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
BSDLLCTest
MoreIsBetter
MoreSCF
QISA
UIElementInspector

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.

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

Related Sample Code
HID Config Save
ImageClient
MoreIsBetter

MoreSCF
QISA

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.

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

Related Sample Code
BSDLLCTest
GrabBag
MoreIsBetter
MoreSCF
QISA

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 7) 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 9) in a `CFArrayCallBacks` (page 19) structure. This callback is used by the `CFCopyDescription` 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 9) in a `CFArrayCallBacks` (page 19) 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 9) in a `CFArrayCallBacks` (page 19) 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 9) in a `CFArrayCallBacks` (page 19) 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 18) 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 17) 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 16) 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 17) 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 some predefined callbacks for your convenience.

```
const CFArrayCallBacks kCFTypeArrayCallBacks;
```

Constants

`kCFTypeArrayCallBacks`

Predefined `CFArrayCallBacks` (page 19) 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`.

Document Revision History

This table describes the changes to *CFArray Reference*.

Date	Notes
2007-05-22	Corrected typographical errors.
2005-12-06	Made minor changes to text to conform to consistency guidelines.
2005-04-29	Moved Introduction to new Introduction page.
2004-10-05	Corrected type for parameters to `CFArrayCreate` (page 9) and `CFArrayGetValues` (page 15).
2003-10-20	Added note to highlight use of `{0,N}` to specify range of entire array.
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