# CFAllocator Reference

**Core Foundation**

**2006-12-08**

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



Simultaneously published in the United States and Canada.

# Contents

# CFAllocator Reference

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

## Overview

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

## Functions by Task

### Creating an Allocator

`CFAllocatorCreate` (page 7)
Creates an allocator object.

### Managing Memory with an Allocator

`CFAllocatorAllocate` (page 6)
Allocates memory using the specified allocator.

`CFAllocatorDeallocate` (page 7)
Deallocates a block of memory with a given allocator.

`CFAllocatorGetPreferredSizeForSize` (page 9)
Obtains the number of bytes likely to be allocated upon a specific request.

`CFAllocatorReallocate` (page 10)
Reallocates memory using the specified allocator.

### Getting and Setting the Default Allocator

`CFAllocatorGetDefault` (page 9)
Gets the default allocator object for the current thread.

`CFAllocatorSetDefault` (page 11)
Sets the given allocator as the default for the current thread.

### Getting an Allocator's Context

`CFAllocatorGetContext` (page 8)
Obtains the context of the specified allocator or of the default allocator.

### Getting the CFAllocator Type ID

`CFAllocatorGetTypeID` (page 10)
Returns the type identifier for the CFAllocator opaque type.

## Functions

### CFAllocatorAllocate

Allocates memory using the specified allocator.

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

**Parameters**

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

*size*
The size of the memory to allocate.

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

**Return Value**
A pointer to the newly allocated memory.

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

**Related Sample Code**
Carbon Porting Tutorial
MoreIsBetter
MoreSCF
QISA

**Declared In**
`CFBase.h`

## CFAllocatorCreate

Creates an allocator object.

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

**Parameters**

*allocator*

The existing allocator to use to allocate memory for the new allocator. Pass the `kCFAllocatorUseContext` (page 20) constant for this parameter to allocate memory using the appropriate function callback specified in the `context` parameter. Pass `NULL` or kCFAllocatorDefault (page 19) to allocate memory for the new allocator using the default allocator.

*context*

A structure of type `CFAllocatorContext` (page 17). The fields of this structure hold (among other things) function pointers to callbacks used for allocating, reallocating, and deallocating memory.

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

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

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

**Declared In**
`CFBase.h`

## CFAllocatorDeallocate

Deallocates a block of memory with a given allocator.

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

**Parameters**

*allocator*

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

*ptr*

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

**Discussion**

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

**Special Considerations**

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

**Availability**

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

**Related Sample Code**

MoreIsBetter

MoreSCF

QISA

**Declared In**

`CFBase.h`

## CFAllocatorGetContext

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

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

**Parameters**

*allocator*

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

*context*

On return, contains the context of *`allocator`*.

**Discussion**

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

**Availability**

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

**Declared In**
CFBase.h

## CFAllocatorGetDefault

Gets the default allocator object for the current thread.

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

**Return Value**
A reference to the default allocator for the current thread. If none has been explicitly set, returns the generic system allocator, kCFAllocatorSystemDefault (page 19). Ownership follows the Get Rule.

**Discussion**
See the discussion for CFAllocatorSetDefault (page 11) for more detail on the default allocator and for advice on how and when to set a custom allocator as the default.

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

**Related Sample Code**
BasicDataBrowser

**Declared In**
CFBase.h

## CFAllocatorGetPreferredSizeForSize

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

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

**Parameters**

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

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

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

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

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

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

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

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

**Declared In**
`CFBase.h`

## CFAllocatorGetTypeID

Returns the type identifier for the CFAllocator opaque type.

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

**Return Value**
The type identifier for the CFAllocator opaque type.

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

**Declared In**
`CFBase.h`

## CFAllocatorReallocate

Reallocates memory using the specified allocator.

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

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

*ptr*

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

*newsize*

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

*hint*

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

**Discussion**

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

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

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

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

**Availability**

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

**Declared In**

`CFBase.h`

## CFAllocatorSetDefault

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

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

**Parameters**

*allocator*

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

**Discussion**

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

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

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

**Availability**

Available in CarbonLib v1.0 and later.

Available in Mac OS X v10.0 and later.

**Declared In**

`CFBase.h`

# Callbacks

## CFAllocatorAllocateCallBack

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

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

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

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

**Parameters**

*allocSize*

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

*hint*

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

*info*

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

**Return Value**
A pointer to the start of the block.

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

**Declared In**
`CFBase.h`

## CFAllocatorCopyDescriptionCallBack

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

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

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

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

**Parameters**

*info*

An untyped pointer to program-defined data.

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

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

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

**Declared In**
`CFBase.h`

## CFAllocatorDeallocateCallBack

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

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

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

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

**Parameters**

*ptr*

The block of memory to deallocate.

*info*

An untyped pointer to program-defined data.

**Discussion**

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

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`CFBase.h`

## CFAllocatorPreferredSizeCallBack

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

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

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

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

**Parameters**

*size*

The amount of memory requested.

*hint*

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

*info*

An untyped pointer to program-defined data.

**Return Value**

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

**Discussion**

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

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`CFBase.h`

## CFAllocatorReallocateCallBack

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

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

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

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

**Parameters**

*ptr*

The block of memory to resize.

*newsize*

The size of the new allocation.

*hint*

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

*info*

An untyped pointer to program-defined data.

**Return Value**
Pointer to the new block of memory.

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

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

**Declared In**
`CFBase.h`

## CFAllocatorReleaseCallBack

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

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

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

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

**Parameters**

*`info`*

The data to be released.

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

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

**Declared In**
`CFBase.h`

## CFAllocatorRetainCallBack

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

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

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

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

**Parameters**

*info*

The data to be retained.

**Discussion**

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

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`CFBase.h`

# Data Types

## CFAllocatorContext

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

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

**Fields**

`version`

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

`info`

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

`retain`

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

`release`

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

`copyDescription`

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

`allocate`

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

`reallocate`

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

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

  Leave the contents of the old block of memory unchanged up to the lesser of the new or old sizes. If the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur. The `hint` argument is a bitfield that you should currently not use (that is, assign 0). If you set this callback to `NULL` the `CFAllocatorReallocate` (page 10) function returns `NULL` in most cases when it attempts to use this allocator.

`deallocate`

A prototype for a function callback that deallocates a given block of memory. In implementing this function, make the block of memory pointed to by `ptr` available for subsequent reuse by the allocator but unavailable for continued use by the program. The `ptr` parameter cannot be `NULL` and if the `ptr` parameter is not a block of memory that has been previously allocated by the allocator, the results are undefined; abnormal program termination can occur. You can set this callback to `NULL`, in which case the `CFAllocatorDeallocate` (page 7) function has no effect.

`preferredSize`

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

**Discussion**

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

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

**Declared In**
`CFBase.h`

### CFAllocatorRef

A reference to a CFAllocator object.

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

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

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

**Declared In**
`CFBase.h`

## Constants

### Predefined Allocators

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

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

**Constants**

`kCFAllocatorDefault`

This is a synonym for `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorSystemDefault`

Default system allocator.

You rarely need to use this.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

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

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

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

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

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

Available in Mac OS X v10.4 and later.

Declared in `CFBase.h`.

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

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

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

`kCFAllocatorUseContext`
Special allocator argument to `CFAllocatorCreate` (page 7)—it uses the functions given in the context to allocate the allocator.

Available in Mac OS X v10.0 and later.

Declared in `CFBase.h`.

**Declared In**
`CFBase.h`

# Document Revision History

This table describes the changes to *CFAllocator Reference*.

| Date | Notes |
| --- | --- |
| 2006-12-08 | Added definition for kCFAllocatorMallocZone. |
| 2005-12-06 | Made minor changes to text to conform to reference consistency guidelines. |
| 2005-08-11 | Clarified description of the 'hint' argument to CFAllocatorAllocate, CFAllocatorReallocate and CFAllocatorGetPreferredSizeForSize. |
| 2003-01-01 | First version of this document. |

Document Revision History

# Index

## C



## K



## P