Memory Management Policy

The basic model used for memory management in a reference-counted environment is provided by a combination of methods defined in the `NSObject` protocol and a standard method naming convention. The `NSObject` class also defines a method, `dealloc`, that is invoked automatically when an object is deallocated. This article describes all the basic rules you need to know to manage memory correctly in a Cocoa program, and provides some examples of correct usage.

Basic Memory Management Rules

The memory management model is based on object ownership. Any object may have one or more owners. As long as an object has at least one owner, it continues to exist. If an object has no owners, the runtime system destroys it automatically. To make sure it is clear when you own an object and when you do not, Cocoa sets the following policy:

- **You own any object you create**

  You create an object using a method whose name begins with “alloc”, “new”, “copy”, or “mutableCopy” (for example, `alloc`, `newObject`, or `mutableCopy`).

- **You can take ownership of an object using retain**

  A received object is normally guaranteed to remain valid within the method it was received in, and that method may also safely return the object to its invoker. You use `retain` in two situations: (1) In the implementation of an accessor method or an `init` method, to take ownership of an object you want to store as a property value; and (2) To prevent an object from being invalidated as a side-effect of some other operation (as explained in Avoid Causing Deallocation of Objects You’re Using).

- **When you no longer need it, you must relinquish ownership of an object you own**

  You relinquish ownership of an object by sending it a `release` message or an `autorelease` message. In Cocoa terminology, relinquishing ownership of an object is therefore typically referred to as “releasing” an object.

- **You must not relinquish ownership of an object you do not own**

  This is just corollary of the previous policy rules, stated explicitly.

A Simple Example

To illustrate the policy, consider the following code fragment:

```
{
    Person *aPerson = [[Person alloc] init];
    // ...
    NSString *name = aPerson.fullName;
    // ...
    [aPerson release];
}
```

The Person object is created using the `alloc` method, so it is subsequently sent a `release` message when it is no longer needed. The person’s name is not retrieved using any of the owning methods, so it is not sent a `release` message. Notice, though, that the example uses `release` rather than `autorelease`.

Use autorelease to Send a Deferred release

You use `autorelease` when you need to send a deferred `release` message—typically when returning an object from a method. For example, you could implement the `fullName` method like this:

```
- (NSString *)fullName {
    NSString *string = [[[NSString alloc] initWithFormat:@"%@ %@",
                                        self.firstName, self.lastName]
autorelease];
    return string;
}
```

You own the string returned by `alloc`. To abide by the memory management rules, you must relinquish ownership of the string before you lose the reference to it. If you use `release`, however, the string will be deallocated before it is returned (and the method would return an invalid object). Using `autorelease`, you signify that you want to relinquish ownership, but you allow the caller of the method to use the returned string before it is deallocated.

You could also implement the `fullName` method like this:

```
- (NSString *)fullName {
    NSString *string = [NSString stringWithFormat:@"%@ %@",
                                self.firstName, self.lastName];
    return string;
}
```

Following the basic rules, you don't own the string returned by `stringWithFormat:`, so you can safely return the string from the method.

By way of contrast, *the following implementation is wrong*:

```
- (NSString *)fullName {
    NSString *string = [[NSString alloc] initWithFormat:@"%@ %@",
                                        self.firstName, self.lastName];
    return string;
}
```

According to the naming convention, there is nothing to denote that the caller of the `fullName` method owns the returned string. The caller therefore has no reason to release the returned string, and it will thus be leaked.

You Don't Own Objects Returned by Reference

Some methods in Cocoa specify that an object is returned by reference (that is, they take an argument of type `ClassName **` or `id *`). A common pattern is to use an `NSError` object that contains information about an error if one occurs, as illustrated by `initWithContentsOfURL:options:error:` (`NSData`) and `initWithContentsOfFile:encoding:error:` (`NSString`).

In these cases, the same rules apply as have already been described. When you invoke any of these methods, you do not create the `NSError` object, so you do not own it. There is therefore no need to release it, as illustrated in this example:

```
NSString *fileName = <#Get a file name#>;
NSError *error;
NSString *string = [[NSString alloc] initWithContentsOfFile:fileName
                        encoding:NSUTF8StringEncoding error:&error];
```

```
if (string == nil) {
    // Deal with error...
}
// ...
[string release];
```

Implement dealloc to Relinquish Ownership of Objects

The `NSObject` class defines a method, `dealloc`, that is invoked automatically when an object has no owners and its memory is reclaimed—in Cocoa terminology it is "freed" or "deallocated.". The role of the `dealloc` method is to free the object's own memory, and to dispose of any resources it holds, including ownership of any object instance variables.

The following example illustrates how you might implement a `dealloc` method for a Person class:

```
@interface Person : NSObject
@property (retain) NSString *firstName;
@property (retain) NSString *lastName;
@property (assign, readonly) NSString *fullName;
@end

@implementation Person
// ...
- (void)dealloc
    [_firstName release];
    [_lastName release];
    [super dealloc];
}
@end
```

Important: Never invoke another object's `dealloc` method directly.

You must invoke the superclass's implementation at the *end* of your implementation.

You should not tie management of system resources to object lifetimes; see Don't Use dealloc to Manage Scarce Resources.

When an application terminates, objects may not be sent a `dealloc` message. Because the process's memory is automatically cleared on exit, it is more efficient simply to allow the operating system to clean up resources than to invoke all the memory management methods.

Core Foundation Uses Similar but Different Rules

There are similar memory management rules for Core Foundation objects (see *Memory Management Programming Guide for Core Foundation*). The naming conventions for Cocoa and Core Foundation, however, are different. In particular, Core Foundation's Create Rule (see The Create Rule) does not apply to methods that return Objective-C objects. For example, in the following code fragment, you are *not* responsible for relinquishing ownership of `myInstance`:

```
MyClass *myInstance = [MyClass createInstance];
```