

The creation, retention and disposal of objects follow a few common patterns.

- Define the object.

```
AClass * myObject;
```

This can occur in the interface of a class definition (called an instance variable) or in a body of code (we'll call it a local variable). Objects are always addressed as pointers, so the `*` is always present.

If it is defined as an instance variable in a class you should do the following.

- Write a property declaration in the interface.

```
@property (nonatomic, retain) AClass* myObject;
```

nonatomic says you don't care about protection from threading problems. That is usually the case, and it is more efficient.

The most common choices are *retain*, *copy* or *assign*. These apply only to the setter.

assign simply moves the value into the object. It overwrites whatever was there. It is used chiefly for scalar types like `int` and `float`.

retain first releases the object that was there, then moves the object pointer and adds a retain. This is used for most object assignments because it safely releases the previous value and retains the new value. A test is made to see whether the object being assigned is the same as the existing object. If so, nothing is done. A release followed by a retain on the same object could destroy the object before its assignment.

copy will release the existing object and then copy the object being assigned. This creates a new object which is given a retain. Copying is done by a method defined for each class. See the `NSCopying` protocol. Copying is usually not needed and is demanding on time and memory. For immutable objects it is pointless unless you want a mutable copy of an immutable object.

To make the variable read-only, add the property *readonly*.

- In the implementation file, "synthesize" it,

```
@synthesize myObject;
```

This will generate the accessors for the object. If it has the readonly property, no setter will be generated. An attempt to set this object will cause the linker to report an error.

- Initialize it.

```
[[AClass alloc] init....];
```

This asks for memory and then calls an init routine to create a valid object in that space. It retains the object and returns a pointer to it.

or,

```
[AClass factoryMethod];
```

This calls a class method which allocates and initializes the object and returns a pointer. If the name of the method begins with “alloc” or “init” or contains “copy” the object is retained. Otherwise it is retained and marked for autorelease.

An object marked for autorelease is placed in an autorelease pool. It behaves exactly like any other object, but when the pool is “drained” a release is sent to each object in the pool. Those whose retain count is 1 at that point will be deallocated. Draining the pool is part of destroying the pool (dealloc). Those objects with a retain count greater than 1 will see it reduced by one, but the pool is gone, so they are not subject to a further release from the pool. The event loop maintains an autorelease pool. Objects you create and autorelease will be disposed of before the next event is processed.

You might want to establish an autorelease pool within part of the application. It is done just as in “main”. Create it before entering the code area, release it when exiting. Pools are stacked. The most recently created is active, those below it are inactive until the pool above is released.

- Put it in an instance variable. Doing this directly avoids using the setter method. It must be done for readonly variables, and if the setter does a retain, this avoids increasing the retain count one more than necessary. All other accesses will follow the rules declared in the @property list.

```
myObject = [[AClass alloc] init....];
```

- Or, put it in a locally defined object in a block of code. @property and @synthesize are not used. No accessors are generated for this object.

```
AClass* myObject = [[AClass alloc] init....];
```

- If it needs a delegate, set it. Usually it will be self.

```
[myObject setDelegate:self];
```

•Write the delegate methods. They must be in the delegate class. If that is self, then the methods probably belong in the file you are editing.

• If the object is a control it may need a target/action setting.

```
[myObject addTarget:self  
    action:@selector(buttonHit:)  
    forControlEvents:UIControlEventTouchUpInside];
```

•Write the action method.

```
-(void) buttonHit:(id) sender  
{.....}
```

•You might put the object into a mutable dictionary, array or set or make it a subview.

```
[aView addSubview:myObject];
```

or,

```
[aSet addObject: myObject];
```

If so, then release it.

```
[myObject release];
```

Most containers like these use setters that specify “retain”. Whenever that is true, it is safe and proper to release the object after the setting. The dictionary or view that owns it will dispose of it at the proper time.

• If the object is an instance variable and has been retained, you should release it in the class “dealloc” method.

```
[myObject release];
```