NSAutoreleasePool

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: Foundation/NSAutoreleasePool.h

Class Description

The NSAutoreleasePool class is used to implement the Foundation Kit's autorelease mechanism. An NSAutoreleasePool object simply contains objects that have received an **autorelease** message, and when deallocated sends a **release** message to each of those objects. An object can be put into the same pool several times, and receives a **release** message for each time it was put into the pool. Thus, sending **autorelease** instead of **release** to an object extends the lifetime of that object until the pool itself is released or longer if the object is retained. This class specification contains information on fine-tuning your application's handling of autorelease pools; see "Object Ownership and Automatic Disposal" in the introduction to the Foundation Kit for general information on using the autorelease mechanism.

You create an NSAutoreleasePool with the usual **alloc** and **init** messages, and dispose of it with **release** (an exception will be raised if you send **autorelease** or **retain** to an autorelease pool). An autorelease pool should always be released in the same context (invocation of a method or function, or body of a loop) that it was created.

NSAutoreleasePools are automatically created and destroyed in applications based on the Application Kit, so your code normally doesn't have to worry about them. (The Application Kit creates a pool at the beginning of the event loop and releases it at the end). There are two cases, though, where you might wish to create and destroy your own autorelease pools. If you're writing a program that's not based on the Application Kit, such as a UNIX tool, there's no built-in support for autorelease pools; you must create and destroy them yourself. Also, if you write a loop that creates many temporary objects, you might wish to create an NSAutoreleasePool inside the loop to dispose of those objects before the next iteration.

Enabling the autorelease mechanism in a program that's not based on the Application Kit is easy. Many programs have a top-level loop where they do most of their work. To enable the autorelease mechanism you create an NSAutoreleasePool at the beginning of this loop and release it at the end. An **autorelease** message sent in the body of the loop automatically puts its receiver into this pool.

Your **main()** function might look like this:

```
void main()
{
   NSArray *args = [[NSProcessInfo processInfo] arguments];
   unsigned count, limit = [args count];

   for (count = 1; count < limit; count++){
      NSAutoreleasePool *pool =[[NSAutoreleasePool alloc] init];
      NSString *fileContents;
      NSString *fileName;

      fileName = [args objectAtIndex:count];
      fileContents = [[NSString alloc] initWithContentsOfFile:fileName];
      [fileContents autorelease];

      /* Process the file, creating and autoreleasing more objects. */

      [pool release];
   }

   /* Do whatever cleanup is needed. */

   exit (EXIT_SUCCESS);
}
```

This program processes files passed in on the command line. The **for** loop processes one file at a time. An NSAutoreleasePool is created at the beginning of this loop and released at the end. Therefore, any object sent an **autorelease** message inside the **for** loop, such as **fileContents**, is added to **pool**, and when **pool** is released at the end of the loop those objects are also released.

Similarly, NSAutoreleasePools can be created inside any loop, even in a program based on the Application Kit, that creates and releases many objects during each iteration.

Nesting Autorelease Pools

Autorelease pools can be nested allowing you to create them in any function or method. For example, the **main()** function may create an autorelease pool and call another function that creates another autorelease pool.

Because an NSAutoreleasePool adds itself to the active pool when created, it doesn't cause a memory leak in the face of an exception or other sudden transfer out of the current context. For example, if an NSAutoreleasePool is created at the beginning of a loop and the program breaks out of the loop during execution, that pool is released by the application's default pool (or whatever pool was active before it was created). In the case of a raised exception, the pool is released when the exception handler's pool is released. Thus it is not necessary to add exception handler code to release objects that were autoreleased.

Guaranteeing the Foundation Ownership Policy

By creating an NSAutoreleasePool instead of simply releasing objects, you extend the lifetime of temporary objects to the lifetime of that pool. After an NSAutoreleasePool is deallocated, you should regard any object that was autoreleased while that pool was active as "disposed of", and not send a message to that object or return it to the invoker of your method.

If you must use a temporary object beyond an autorelease context, you can do so by sending a **retain** message to the object within the context and then send it **autorelease** after the pool has been released as in:

```
- findMatchingObject:anObject
{
    id match = nil;

    while (match == nil) {
        NSAutoreleasePool *subpool = [[NSAutoreleasePool alloc] init];

        /* Do a search that creates a lot of temporary objects. */
        match = [self expensiveSearchForObject:anObject];

        if (match != nil) [match retain]; /* Keep match around. */

        [subpool release];
    }

    return [match autorelease];   /* Let match go and return it. */
}
```

By sending **retain** to **match** while **subpool** is in effect and sending **autorelease** to it after **subpool** has been released, **match** is effectively moved from **subpool** to the pool that was previously active. This extends the lifetime of **match** and allows it to receive messages outside the loop and be returned to the invoker of **findMatchingObject:**.

Method Types

Adding an object to the current pool	+ addObject:
Adding an object	– addObject:
Debugging autorelease mechanism	+ enableDoubleReleaseCheck: + enableRelease: + resetTotalAutoreleasedObjects + setPoolCountThreshold: + showPools + showPoolsWithObjectIdenticalTo: + totalAutoreleasedObjects

Class Methods

addObject:

+ (void)**addObject:**(id)*anObject*

Adds *anObject* to the active autorelease pool in the current thread, so that it will be sent a **release** message when the pool itself is deallocated. The same object may be added several times to the active pool and will receive a **release** message for each time it was added. Normally you don't invoke this method directly—send **autorelease** to *anObject* instead.

See also: – addObject:

enableDoubleReleaseCheck:

+ (void)**enableDoubleReleaseCheck:**(BOOL)*enable*

This method aids in debugging the autorelease mechanism. If *enable* is YES, the **release** and **autorelease** methods check to see if the receiving object has been released too many times. This check is performed by searching all pools; consequently, programs may run very slow. Double release check is disabled by default.

enableRelease:

+ (void)**enableRelease:**(BOOL)*enable*

This method aids in debugging the autorelease mechanism. If *enable* is NO, **release** and **autorelease** messages are effectively ignored, allowing all objects to remain in memory. Note that this will cause your application's use of memory to increase. Release is enabled by default.

resetTotalAutoreleasedObjects

+ (void)**resetTotalAutoreleasedObjects**

This method aids in debugging the autorelease mechanism. Use **totalAutoreleasedObjects** to return the number of all autoreleased objects since the last invocation of this method or from the beginning of program execution, if this method was never invoked. Use this method to start a new count of autoreleased objects, beginning with the next object that is sent **autorelease**.

setPoolCountThreshold:

+ (void)**setPoolCountThreshold:**(unsigned)*count*

This method aids in debugging the autorelease mechanism. When the pool size reaches a multiple of *count*, this method will call a well-known method (indicated in the console). You can then set a breakpoint on that method in the debugger. If count is 0, this feature is disabled. The default setting is 0.

showPools

+ (void)**showPools**

This method aids in debugging the autorelease mechanism by printing to **stdout** a description of all autorelease pools.

See also: + **showPoolsWithObjectIdenticalTo:**

showPoolsWithObjectIdenticalTo:

+ (void)**showPoolsWithObjectIdenticalTo:**(id)*anObject*

This method aids in debugging the autorelease mechanism by printing to **stdout** a description of all autorelease pools containing *anObject*. Containment is determined by comparing object **id**s.

See also: + **showPools**

totalAutoreleasedObjects

+ (unsigned)**totalAutoreleasedObjects**

This method aids in debugging the autorelease mechanism. Invoke this method to return the number of all autoreleased objects since the last invocation of **resetTotalAutoreleasedObjects** or from the beginning of program execution, if **resetTotalAutoreleasedObjects** was never invoked. Use **resetTotalAutoreleasedObjects** to start a new count of autoreleased objects, beginning with the next object that is sent **autorelease**.

Instance Methods

addObject:

– (void)**addObject:**(id)*anObject*

Adds *anObject* to the receiver, so that it will be sent a **release** message when the receiver is deallocated. The same object may be added several times to the same pool and will receive a **release** message for each time it was added. Normally you don't invoke this method directly—send **autorelease** to *anObject* instead.

See also: + **addObject:**