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. When a thread terminates, it automatically releases all of the autorelease pools associated with itself.

Autorelease pools can be nested, so you can include them in any function or method. For example, a **main()** function may create an autorelease pool and call another function that creates another autorelease pool. Or a single method might have an autorelease pool for an outer loop, and another autorelease pool for an inner loop. Each thread in a program maintains autorelease pools on a stack; the most recently created (and unreleased) autorelease pool is the top pool on the stack. The ability to nest autorelease pools is a definite advantage, but there are side effects when exceptions occur (see "Exceptions and Nested Autorelease Pools").

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.

Implications of Nested Autorelease Pools

It’s common to speak of autorelease pools as being nested because of the enclosure evident in code; for instance, you can have an autorelease pool in an outer loop that contains an inner loop with its own autorelease pool. But you can also think of nested autorelease pools as being on a stack, with the "inmost" autorelease pool being on top of the stack. As noted earlier, this is actually how nested autorelease pools are implemented: Each thread (NSThread) in a program maintains a stack of autorelease pools. When you create an autorelease pool, it is pushed onto the top of the current thread’s stack. Autoreleased objects—that is, objects which have received an **autorelease** message or which are added through the **addObject:** class method—are always put in the autorelease pool at the top of the stack.

Released autorelease pools, if not on the top of the stack, will cause all (unreleased) autorelease pools above them on the stack to be released, along with all their objects. If you neglect to send **release** to an autorelease pool when you're finished with it (something *not* recommended), it will be released when one of the autorelease pools in which it nests is released.

This behavior has implications for exceptional conditions. If an exception occurs, and the thread suddenly transfers out of the current context, the pool associated with that context is released. However, if that pool is not the top pool on the thread's stack, all the pools above the released pool are also released (releasing all their objects in the process). The top autorelease pool on the thread's stack then becomes the pool previously underneath the released pool associated with the exceptional condition. Because of this behavior, exception handlers do not need to release objects that were sent **autorelease**. Neither is it necessary or even desireable for an exception handler to send **release** to its autorelease pool, unless the handler is re-raising the exception.

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:**.

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:**

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:**