NSObject

Inherits From:	none *(NSObject is the root class)*
Conforms To:	NSObject
Declared In:	Foundation/NSObject.h Foundation/NSRunLoop.h

Class Description

NSObject is the root class of all ordinary Objective C inheritance hierarchies; it has no superclass. Its interface derives from two sources: the methods it declares directly and those declared in the NSObject protocol. Its interface is divided in this way so that objects inheriting from other root classes (notably NSProxy) can stand in for ordinary objects without having to inherit from NSObject. The following discussion makes no distinction between the methods declared in this class and those declared in the NSObject protocol.

From NSObject, other classes inherit a basic interface to the run-time system for the Objective C language. It's through NSObject that instances of all classes obtain their ability to behave as objects. Among other things, the NSObject class provides inheriting classes with a framework for creating, initializing, deallocating, comparing, and archiving objects, for performing methods selected at run-time, for querying an object about its methods and its position in the inheritance hierarchy, and for forwarding messages to other objects. For example, to ask an object what class it belongs to, you'd send it a **class** message. To find out whether it implements a particular method, you'd send it a **respondsToSelector:** message

The NSObject class is an abstract class; programs use instances of classes that inherit from NSObject, but never of NSObject itself.

Initializing an Object to Its Class

Every object is connected to the run-time system through its **isa** instance variable, inherited from the NSObject class. **isa** identifies the object's class; it points to a structure that's compiled from the class definition. Through **isa**, an object can find whatever information it needs at run timeÐsuch as its place in the inheritance hierarchy, the size and structure of its instance variables, and the location of the method implementations it can perform in response to messages.

Because all ordinary objects inherit directly or indirectly from the NSObject class, they all have this variable. The defining characteristic of an ªobjectº is that its first instance variable is an **isa** pointer to a class structure.

The installation of the class structureÐthe initialization of **isa**Ðis one of the responsibilities of the **alloc** and **allocWithZone:** methods, the same methods that create (allocate memory for) new instances of a class. In other words, class initialization is part of the process of creating an object; it's not left to the methods, such as **init**, that initialize individual objects with their particular characteristics.

Instance and Class Methods

Every object requires an interface to the run-time system, whether it's an instance object or a class object. For example, it should be possible to ask either an instance or a class whether it can respond to a particular message. So that this won't mean implementing every NSObject method twice, once as an instance method and again as a class method, the run-time system treats methods defined in the root class in a special way:

> *Instance methods defined in the root class can be performed both by instances*
> *and by class objects.*

A class object has access to class methodsÐthose defined in the class and those inherited from the classes above it in the inheritance hierarchyÐbut generally not to instance methods. However, the run-time system gives all class objects access to the instance methods defined in the root class. Any class object can perform any root instance method, provided it doesn't have a class method with the same name.

For example, a class object could be sent messages to perform NSObject's **respondsToSelector:** and **perform:withObject:** instance methods:

```
SEL method = @selector(riskAll:);

if ( [MyClass respondsToSelector:method] )
    [MyClass perform:method withObject:self];
```

When a class object receives a message, the run-time system looks first at the receiver's set of class methods. If it fails to find a class method that can respond to the message, it looks at the set of instance methods defined in the root class. If the root class has an instance method that can respond (as NSObject does for **respondsToSelector:** and **perform:withObject:**), the run-time system uses that implementation and the message succeeds.

Note that the only instance methods available to a class object are those defined in the root class. If MyClass in the example above had reimplemented either **respondsToSelector:** or **perform:withObject:**, those new versions of the methods would be available only to instances. The class object for MyClass could perform only the versions defined in the NSObject class. (Of course, if MyClass had implemented **respondsToSelector:** or **perform:withObject:** as class methods rather than instance methods, the class would perform those new versions.)

Initializing the Class

+ (void)**initialize**	Initializes the class before it's used (before it receives its first message).

Creating and Destroying Instances

+ (id)**alloc**	Returns a new, uninitialized instance of the receiving class.
+ (id)**allocWithZone:**(NSZone *)*zone*	Returns a new, uninitialized instance of the receiving class in *zone*.
+ (id)**new**	Allocates a new instance of the receiving class, sends it an **init** message, and returns the initialized object returned by **init**. This method is simply a convenient cover for the **alloc** and **init** methods.
- (id)**copy**	Invokes **copyWithZone:**. This method is implemented in NSObject as a convenience to subclasses. A subclass need override only **copyWithZone:** for both **copy** and **copyWithZone:** to operate correctly.
- (void)**dealloc**	Deallocates the memory occupied by the receiver.
- (id)**init**	Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.
- (id)**mutableCopy**	Invokes **mutableCopyWithZone:**. This method is implemented in NSObject as a convenience to subclasses. A subclass need override only **mutableCopyWithZone:** for both **mutableCopy** and **mutableCopyWithZone:** to operate correctly.

Identifying Classes

+ (Class)**class**	Returns **self**. Since this is a class method, it returns the class object.
+ (Class)**superclass**	Returns the class object for the receiver's superclass.

Testing Class Functionality

+ (BOOL)**instancesRespondToSelector:**(SEL)*aSelector*	Returns YES if instances of the class are capable of responding to *aSelector* messages, and NO if they're not.

Testing Protocol Conformance

+ (BOOL)**conformsToProtocol:**(Protocol *)*aProtocol*	Returns YES if the receiving class conforms to *aProtocol*, and NO if it doesn't.

Obtaining Method Information

+ (IMP)**instanceMethodForSelector:**(SEL)*aSelector*	Locates and returns the address of the implementation of the *aSelector* instance method.
- (IMP)**methodForSelector:**(SEL)*aSelector*	Locates and returns the address of the receiver's implementation of the *aSelector* method, so that it can be called as a function.
- (NSMethodSignature *)**methodSignatureForSelector:**(SEL)*aSelector*	Returns an object that contains a description of the *aSelector* method, or **nil** if the *aSelector* method can't be found.

Describing Objects

+ (NSString *)**description**	Subclasses override this method to return a human-readable string representation of

the contents of the receiver. NSObject's implementation simply prints the name of the receiver's class.

Posing

+ (void)**poseAsClass:**(Class)*aClass*

Causes the receiving class to ªpose asº its superclass.

Error Handling

- (void)**doesNotRecognizeSelector:**(SEL)*aSelector*

Handles *aSelector* messages that the receiver doesn't recognize.

Sending Deferred Messages

+ (void)**cancelPreviousPerformRequestsWithTarget:**(id)*aTarget* **selector:**(SEL)*aSelector* **object:**(id)*anObject*

Cancels previous perform requests having the same target and argument (as determined by **isEqual:**), and the same selector. This method removes timers only in the current run loop, not all run loops.

- (void)**performSelector:**(SEL)*aSelector* **object:**(id)*anObject* **afterDelay:**(NSTimeInterval)*delay*

Sends an *aSelector* message to *anObject* after *delay*. **self** and *anObject* are retained until after the action is executed.

Forwarding Messages

- (void)**forwardInvocation:**(NSInvocation *)*anInvocation*

Implemented by subclasses to forward messages to other objects.

Archiving

- (id)**awakeAfterUsingCoder:**(NSCoder *)*aDecoder*

Implemented by subclasses to reinitialize the receiver. The NSObject implementation of this method simply returns **self**.

- (Class)**classForArchiver**	Identifies the class to be used during archiving. NSObject's implementation returns the object returned by **classForCoder:**.
- (Class)**classForCoder**	Identifies the class to be used during serialization. An NSObject returns its own class by default.
- (id)**replacementObjectForArchiver:**(NSArchiver *)*anArchiver*	Allows an object to substitute another object for itself during archiving. NSObject's implementation returns the object returned by **replacementObjectForCoder:**.
- (id)**replacementObjectForCoder:**(NSCoder *)*anEncoder*	Allows an object to substitute another object for itself during serialization. NSObject's implementation returns **self**.
+ (void)**setVersion:**(int)*version*	Sets the class version number to *version*.
+ (int)**version**	Returns the version of the class definition.