NSObject

Adopted By: NSObject

Declared In: foundation/NSObject.h

Protocol Description

The NSObject protocol groups methods that are fundamental to all instances. If an object conforms to this protocol, it can be considered a first-class object in NEXTSTEP. Such an object can be asked about its:

- Class, and the place of its class in the inheritance hierarchy
- Conformance to protocols
- Ability to respond to a particular message

In addition, objects that conform to this protocol—with its **retain**, **release**, and **autorelease** methods—can also integrate with the object-management and deallocation scheme defined in the Foundation Kit. (See the introduction to the Foundation Kit for more information.) Thus, an object that conforms to the NSObject protocol can be managed by container objects like those defined by NSArray and NSDictionary.

NEXTSTEP's root class, NSObject, adopts this protocol, so virtually all objects in NEXTSTEP have the features described by this protocol.

Method Types

Identifying and comparing instances	– isEqual: – hash – self
Identifying class and superclass	– class – superclass
Determining allocation zones	– zone

Sending messages determined at run time	– perform: – perform:withObject: – perform:withObject:withObject:
Identifying proxies	– isProxy
Testing inheritance relationships	– isKindOfClass: – isMemberOfClass:
Testing for protocol conformance	– conformsToProtocol:
Testing class functionality	– respondsToSelector:
Managing reference counts	– retain – release – autorelease – retainCount

Instance Methods

autorelease

– **autorelease**

Adds the receiver to the current autorelease pool and returns **self**. You add an object to an autorelease pool so that it receives a **release** message—and thus might be deallocated—when the pool is destroyed. For more information on the autorelease mechanism, see the NSAutoreleasePool class specification.

See also: – **retain**, – **release**

class

– (Class)**class**

Returns the class object for the receiver's class.

See also: + **class** (NSObject class)

conformsToProtocol:

– (BOOL)**conformsToProtocol:**(Protocol *)*aProtocol*

Returns YES if the class of the receiver conforms to *aProtocol*, and NO if it doesn't. This method works identically to the **conformsToProtocol:** class method declared in NSObject. It's provided as a convenience so that you don't need to get the class object to find out whether an instance can respond to a given set of messages.

See also: + **conformsToProtocol:** (NSObject class)

hash

– (unsigned int)**hash**

Returns an unsigned integer that can be used as a table address in a hash table structure. If two objects are equal (as determined by the **isEqual:** method), they must have the same hash value.

See also: – **isEqual:**

isEqual:

– (BOOL)**isEqual:***anObject*

Returns YES if the receiver and *anObject* are equal; otherwise returns NO. For the NSObject class, the **id** of *anObject* and the receiver are compared to determine equality. Subclasses can override this method to redefine what it means for instance to be equal. For example, a container object might define two containers as equal if they contain the same contents, even though the contains themselves are different objects. See the NSData, NSDictionary, NSArray, and NSString class specifications for examples of the use of this method.

isKindOfClass:

– (BOOL)**isKindOfClass:**(Class)*aClass*

Returns YES if the receiver is an instance of *aClass* or an instance of any class that inherits from *aClass*. Otherwise, it returns NO. For example, in this code **isKindOfClass:** would return YES because, in the Application Kit, the Menu class inherits from Window:

```
id aMenu = [[Menu alloc] init];
if ( [aMenu isKindOfClass:[Window class]] )
    . . .
```

When the receiver is a class object, this method returns YES if *aClass* is the NSObject class, and NO otherwise.

See also: – **isMemberOfClass:**

isMemberOfClass:

– (BOOL)**isMemberOfClass:**(Class)*aClass*

Returns YES if the receiver is an instance of *aClass*. Otherwise, it returns NO. For example, in this code, **isMemberOfClass:** would return NO:

```
id aMenu = [[Menu alloc] init];
if ([aMenu isMemberOfClass:[Window class]])
    . . .
```

When the receiver is a class object, this method returns NO. Class objects are not "members of" any class.

See also: – **isKindOfClass:**

isProxy

– (BOOL)**isProxy**

Returns YES to indicate that the receiver is an NSProxy, rather than an object that descends from NSObject. Otherwise, it returns NO. NSObject's implementation of this method returns NO to indicate that an NSObject is a normal object, not a stand-in for one.

perform:

– **perform:**(SEL)*aSelector*

Sends an *aSelector* message to the receiver and returns the result of the message. If *aSelector* is null, an NSInvalidArgumentException is raised.

perform: is equivalent to sending an *aSelector* message directly to the receiver. For example, all three of the following messages do the same thing:

```
id myClone = [anObject copy];
id myClone = [anObject perform:@selector(copy)];
id myClone = [anObject perform:sel_getUid("copy")];
```

However, the **perform:** method allows you to send messages that aren't determined until run time. A variable selector can be passed as the argument:

```
SEL myMethod = findTheAppropriateSelectorForTheCurrentSituation();
[anObject perform:myMethod];
```

aSelector should identify a method that takes no arguments. If the method returns anything but an object, the return must be cast to the correct type. For example:

```
char *myClass;
myClass = (char *)[anObject perform:@selector(name)];
```

Casting generally works for pointers and for integral types that are the same size as pointers (such as **int** and **enum**). Whether it works for other integral types (such as **char**, **short**, or **long**) is machine dependent. Casting doesn't work if the return is a floating type (**float** or **double**) or a structure or union. This is because the C language doesn't permit a pointer (like **id**) to be cast to these types.

Therefore, **perform:** shouldn't be asked to perform any method that returns a floating type, structure, or union, and should be used very cautiously with methods that return integral types. An alternative is to get the address of the method implementation (using the **methodForSelector:** method declared by the NSObject class) and call it as a function. For example:

```
SEL aSelector = @selector(backgroundGray);
float aGray = ((float (*)(id, SEL))
        [anObject methodForSelector:aSelector] )(anObject, aSelector);
```

See also: – **perform:withObject:**, – **perform:withObject:withObject:**, – **methodForSelector:** (NSObject class)

perform:withObject:

– **perform:**(SEL)*aSelector* **withObject:***anObject*

Sends an *aSelector* message to the receiver with *anObject* as an argument. If *aSelector* is null, an NSInvalidArgumentException is raised.

This method is the same as **perform:** except that you can supply an argument for the *aSelector* message. *aSelector* should identify a method that takes a single argument of type **id**.

See also: – **perform:**, – **perform:withObject:withObject:**, – **methodForSelector:** (NSObject class)

perform:withObject:withObject:

– **perform:**(SEL)*aSelector*
withObject:*anObject*
withObject:*anotherObject*

Sends the receiver an *aSelector* message with *anObject* and *anotherObject* as arguments. If *aSelector* is null, an NSInvalidArgumentException is raised. This method is the same as **perform:** except that you can supply two arguments for the *aSelector* message. *aSelector* should identify a method that can take two arguments of type **id**.

See also: – **perform:**, – **perform:withObject:**, – **methodForSelector:** (NSObject class)

release

– (void)**release**

As defined in the NSObject class, **release** decrements the receivers's reference count. When an object's reference count reaches 0, the object is automatically deallocated.

You send **release** messages only to objects that you "own". By definition, you own objects that you create using one of the **alloc...** or **copy...** methods. These object creation methods return objects with an implicit reference count of one. You also own (or perhaps share ownership in) an object that you send a **retain** message to, since **retain** increments the object's reference count. Each **retain** message you send an object should be balanced eventually with a **release** message, so that the object can be deallocated. For more information on the automatic deallocation mechanism, see the introduction to the Foundation Kit.

You would only implement this method to define your own reference-counting scheme. Such implementations should not invoke the inherited method; that is, they should not include a **release** message to **super**.

See also: – **autorelease,** – **release**, – **retainCount**

respondsToSelector:

– (BOOL)**respondsToSelector:**(SEL)*aSelector*

Returns YES if the receiver implements or inherits a method that can respond to *aSelector* messages, and NO if it doesn't. The application is responsible for determining whether a NO response should be considered an error.

Note that if the receiver is able to forward *aSelector* messages to another object, it will be able to respond to the message, albeit indirectly, even though this method returns NO.

See also: – **forwardInvocation:** (NSObject class),
+ **instancesRespondToSelector:**(NSObject class)

retain

– **retain**

As defined in the NSObject class, **retain** increments the receiver's reference count. You send an object a **retain** message when you want to prevent it from being deallocated without your express permission.

An object is deallocated automatically when its reference count reaches 0. **retain** messages increment the reference count, and **release** messages decrement it. For more information on this mechanism, see the introduction to the Foundation Kit.

As a convenience, **retain** returns **self**, since it is often used in nested expressions:

```
NSString *headerDir = [[NSString
    stringWithCString:"/LocalLibrary/Headers"] retain]
```

You would only implement this method if you were defining your own reference-counting scheme. Such implementations must return **self** and should not invoke the inherited method by sending a **retain** message to **super**.

See also: – **autorelease,** – **release**, – **retainCount**

retainCount

– (unsigned int)**retainCount**

Returns the receiver's reference count for debugging purposes. You rarely send a **retainCount** message; however, you might implement this method in a class to implement your own reference-counting scheme. For objects that never get released (that is, their **release** method does nothing), this method should return UINT_MAX, as defined in **<limits.h>**.

See also: – **autorelease,** – **release**, – **retain**

self

– **self**

Returns the receiver.

See also: – **class**

superclass

– **superclass**

Returns the class object for the receiver's superclass.

See also: + **superclass** (NSObject class)

zone

– (NSZone *)**zone**

Returns a pointer to the zone from which the receiver was allocated. Objects created without specifying a zone are allocated from the default zone, which is returned by **NSDefaultMallocZone()**.

See also: + **allocWithZone:** (NSObject class)