NSProxy

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

Class Description

NSProxy is an abstract superclass defining an API for objects that act as stand-ins for other objects or for objects that don't exist yet. Typically, a message to a proxy is forwarded to the real object, or causes the proxy to load (or transform itself into) the real object. Subclasses of NSProxy can be used to implement transparent distributed messaging (for example, NSDistantObject) or for lazy instantiation of objects that are expensive to create.

NSProxy implements the basic methods required of a root class, including those defined in the NSObject protocol. However, as an abstract class it doesn't provide an initialization method, and it raises an exception upon receiving any message it doesn't respond to. A concrete subclass must therefore provide an initialization or creation method and override the **forwardInvocation:** and **methodSignatureForSelector:** methods to handle messages that it doesn't implement itself. A subclass's implementation of **forwardInvocation:** should do whatever is needed to process the invocation, such as forwarding the invocation over the network or loading the real object and passing it the invocation. **methodSignatureForSelector:** is required to provide argument type information for a given message; a subclass's implementation should be able to determine the argument types for the messages it needs to forward and should construct an NSMethodSignature accordingly. See the NSDistantObject, NSInvocation, and NSMethodSignature class specifications for more information.

Adopted Protocols

NSObject	– autorelease – class – conformsToProtocol: – description – hash – isEqual: – isKindOfClass: – isMemberOfClass: – isProxy – performSelector: – performSelector:withObject: – performSelector:withObject:withObject: – release – respondsToSelector: – retain – retainCount – self – superclass – zone

Method Types

Creating instances	+ alloc + allocWithZone:
Deallocating instances	– dealloc
Getting the class	+ class
Handling unimplemented methods	– forwardInvocation: – methodSignatureForSelector:
Getting a description	– description

Class Methods

alloc

+ (id)**alloc**

Returns a new instance of the receiving class, as described in the NSObject class specification under the **alloc** class method.

allocWithZone:

+ (id)**allocWithZone:**(NSZone *)*zone*

Returns a new instance of the receiving class, as described in the NSObject class specification under the **allocWithZone:** class method.

class

+ (Class)**class**

Returns **self**. Since this is a class method, it returns the class object.

See also: + **class** (NSObject), – **class** (NSObject protocol)

load

+ (void)**load**

This method is invoked whenever a class or category is added to the Objective-C runtime; implement this method to perform class-specific behavior upon loading. It is sent to classes and categories that are both dynamically loaded and statically linked, but only if the newly-loaded class or category implements a method that can respond. As an example, when Interface Builder loads a palette, the load method is sent to each class and category in the palette.

load is usually invoked before **initialize**. It is usually the very first method sent to the class, although this isn't guaranteed. The order in which classes are loaded is also not guaranteed, to the point that superclasses aren't even guaranteed to be loaded before all of their subclasses. Because you can't rely on other classes being loaded at the point when your class is sent a **load** message, you should be extremely careful when messaging other classes from within your load method.

Warning: Due to the amount of uncertainty about the environment at the point that **load** is invoked, you should avoid using **load** whenever possible. All class-specific initialization should be done in the class's **initialize** method.

Note that although **load** is essentially replaces NEXTSTEP's **finishLoading:** method, the circumstances surrounding their invocation is slightly different. Consult your NEXTSTEP Developer documentation if you are porting code that uses **finishLoading:**.

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

respondsToSelector:

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

Returns YES if the receiving class responds to *aSelector* messages, NO otherwise.

nstance Methods

class

@protocol NSObject
– (Class)**class**

Returns the class of the receiver (not the class of the real object).

conformsToProtocol:

@protocol NSObject
– (BOOL)**conformsToProtocol:**(Protocol)*aProtocol*

Uses **forwardInvocation:** to send the **conformsToProtocol:** message to the real object and returns the result. Note that NSProxy's implementation of **forwardInvocation:** merely raises an exception.

dealloc

– (void)**dealloc**

Deallocates the memory occupied by the receiver, as described in the NSObject class specification under the **dealloc** instance method.

description

– (NSString *)**description**

Returns an NSString containing the real class name and the **id** of the receiver as a hexadecimal number.

forwardInvocation:

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

Passes *anInvocation* on to the real object that the proxy represents. NSProxy's implementation merely raises NSInvalidArgumentException. Override this method in your subclass to handle *anInvocation* appropriately, at the very least by setting its return value.

For example, if your proxy merely forwards messages to an instance variable named **realObject**, it can implement **forwardInvocation:** like this:

```
– (void)forwardInvocation:(NSInvocation *)anInvocation
{
    [anInvocation setTarget:realObject];
    [anInvocation invoke];
    return;
}
```

isKindOfClass:

@protocol NSObject
– (BOOL)**isKindOfClass:**(Class)*aClass*

Uses **forwardInvocation:** to send the **isKindOfClass:** message to the real object and returns the result. Note that NSProxy's implementation of **forwardInvocation:** merely raises an exception.

isMemberOfClass:

@protocol NSObject
– (BOOL)**isMemberOfClass:**(Class)*aClass*

Uses **forwardInvocation:** to send the **isMemberOfClass:** message to the real object and returns the result. Note that NSProxy's implementation of **forwardInvocation:** merely raises an exception.

isProxy

@protocol NSObject
– (BOOL)**isProxy**

Returns YES. Subclasses shouldn't override this method to return NO.

methodSignatureForSelector:

– (NSMethodSignature *)**methodSignatureForSelector:**(SEL)*aSelector*

Raises NSInvalidArgumentException. Override this method in your concrete subclass to return a proper NSMethodSignature for *aSelector* and the class that your proxy objects stand in for. Be sure to avoid an infinite loop when necessary by checking that *aSelector* isn't the selector for this method itself and by not sending any message that might invoke this method.

For example, if your proxy merely forwards messages to an instance variable named **realObject**, it can implement **methodSignatureForSelector:** like this:

```
– (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    return [realObject methodSignatureForSelector:aSelector];
}
```

See also: – **methodSignatureForSelector:** (NSObject)

respondsToSelector:

@protocol NSObject
– (BOOL)**respondsToSelector:**(SEL)*aSelector*

Uses **forwardInvocation:** to send the **respondsToSelector:** message to the real object and returns the result. Note that NSProxy's implementation of **forwardInvocation:** merely raises an exception.