NSNotification

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	Foundation/NSNotification.h

Class Description

NSNotification objects encapsulate information so that it can be broadcast to other objects by an NSNotificationCenter object.

Notifications and their Rationale

The standard way to pass information between objects is message passing—one object invokes the method of another object. However, message passing requires that the object sending the message know who the receiver is and what messages it responds to. At times, this tight coupling of two objects is undesirable—most notably because it would join together two otherwise independent subsystems. For these cases, a broadcast model is introduced: An object posts a notification, which is dispatched to the appropriate observers through an NSNotificationCenter object, or simply *notification center*.

An NSNotification object (referred to as a *notification*) contains a name, an object, and an optional dictionary. The name is a tag identifying the notification. The object is any object that the poster of the notification wants to send to observers of that notification (typically, it is the object that posted the notification). The dictionary stores other related objects if any.

Any object may post a notification. Other objects can register themselves as observers to receive notifications when they are posted. The object posting the notification, the object included in the notification, and the observer of the notification may all be different objects or the same object. Objects that post notifications need not know anything about the observers. On the other hand, observers need to know at least the notification name and keys to the dictionary if provided.

NSNotification objects are immutable objects.

Notification Centers

The notification center manages the sending and receiving of notifications. When an object wants to receive a certain notification, it registers itself with the notification center. When an object has a notification to send, it sends it to the notification center. When the notification center receives a notification, it passes that

notification along to all objects registered to receive it. (See the NSNotificationCenter class specification for more on posting notifications.)

This notification model frees an object from concern about what objects it should send information to. Any object may simply post a notification without knowing what objects—if any—are receiving the notification. However, objects receiving notifications do need to know at least the notification name if not the type of information the notification contains. The notification center takes care of broadcasting notifications to registered observers. Another benefit of this model is to allow multiple objects to listen for notifications, which would otherwise be cumbersome.

You can create a notification object with the class methods **notificationWithName:object:** or **notificationWithName:object:userInfo:**. However, you don't usually create your own notifications directly. The NSNotificationCenter methods **postNotificationName:object:** and **postNotificationName:object:userInfo:** allow you to conveniently post a notification without creating it first.

Notification and Delegation

Using the notification system is similar to using delegates, but it has these advantages:

- Any number of objects may receive the notification, not just the delegate object. This precludes returning a value.
- An object may receive any message you like from the notification center, not just the predefined delegate methods.
- The object posting the notification does not even have to know the observer exists.

NSCopying Protocol

The NSNotification class adopts the NSCopying protocol, making it possible to treat notifications as context-independent values that can be copied and reused. You can store a notification for later use or use the Distributed Objects system to send a notification to another process. The NSCopying protocol essentially allows clients to deal with notifications as first class values that can be copied by collections. You can put notifications in an array and send the **copy** message to that array, which recursively copies every item.

Creating Subclasses

You can subclass NSNotification to contain information in addition to the notification name, object, and dictionary. This extra data must be agreed upon between notifiers and observers.

NSNotification is actually a class cluster. As such, it provides no storage for the name, object, and userInfo values. To subclass NSNotification, you must override the primitive methods **name**, **object**, and **userInfo**. Use the **init** method as the designated initializer.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copyWithZone:

Method Types

Creating a notification	+ notificationWithName:object: + notificationWithName:object:userInfo:
Obtaining information about a notification	– name – object – userInfo

Class Methods

notificationWithName:object:

+ (id)**notificationWithName:**(NSString *)*aName*
object:(id)*anObject*

Returns a notification object that associates the name *aName* with the object *anObject*.

This method copies *aName* and retains *anObject*.

See also: **– postNotificationName:object:** (NSNotificationCenter)

notificationWithName:object:userInfo:

+ (id)**notificationWithName:**(NSString *)*aName*
object:(id)*anObject*
userInfo:(NSDictionary *)*userInfo*

Returns a notification object that associates the name *aName* with the object *anObject* and the dictionary of arbitrary data, *userInfo*. The dictionary *userInfo* may be **nil**.

This method copies *aName* and retains both *anObject* and *userInfo*.

See also: **+ notificationWithName:object:,**
– postNotificationName:object:userInfo: (NSNotificationCenter)

nstance Methods

name

– (NSString *)**name**

Returns the name of the notification. Examples of this might be “PortIsInvalid” or “PhoneRinging.” Typically, you invoke this method on the notification object passed to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

Notification names can be any string. To avoid name collisions, however, you might want to use a prefix that’s specific to your application.

object

– (id)**object**

Returns the object associated with this notification. This is often the object that posted this notification. It may be **nil**.

Typically, you invoke this method on the notification object passed in to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

For example, suppose you’ve registered an object to receive the message **handlePortDeath:** when the “PortInvalid” notification is posted to the notification center and that **handlePortDeath:** needs to access the object monitoring the port that is now invalid. **handlePortDeath:** can retrieve that object as shown here:

```
- (void)handlePortDeath:(NSNotification *)notification
{
    ...
    [self reclaimResourcesForPort:[notification object]];
    ...
}
```

userInfo

– (NSDictionary *)**userInfo**

Returns the NSDictionary associated with this notification or **nil** if there is no such object. The NSDictionary stores any additional objects that objects receiving this notification might use. For example in the Application Kit, NSControl objects post the NSControlTextDidChangeNotification whenever the field editor (an NSText object) changes text inside the NSControl. This notification provides both the NSControl object and the field editor to objects registered to receive it. The field editor is returned when you access the dictionary, as shown here:

```
- (void)controlTextDidBeginEditing:(NSNotification *)notification
{
```

```
    NSText *fieldEditor = [[notification userInfo]
       objectForKey:@"NSFieldEditor"];/* the field editor */
    NSControl *postingObject = [notification object];
       /* The object that posted the notification. */
    ...
}
```