EOObserverCenter

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	EOControl/EOObserver.h

Class Description

EOObserverCenter and related classes form an efficient, specialized mechanism for notification of changes to objects. Observable objects invoke the **willChange** method before altering their state, which causes all registered observers to receive an **objectWillChange:** message. The entire observation mechanism is defined by four classes and a protocol: EOObserverCenter, EODelayedObserverQueue, EODelayedObserver, EOObserverProxy, and EOObserving. EOObserverCenter is the central manager of change notification. It records all observers and the objects they observe, and distributes the **objectWillChange:** message defined by the EOObserving protocol. The other three classes add to the basic observation mechanism. EODelayedObserverQueue alters the basic, synchronous change notification mechanism by offering different priority levels, which allows observers to specify the order in which they're notified of changes. EODelayedObserver is an abstract superclass for objects that observe other objects (such as the Interface layer's EOAssociation classes). Finally, EOObserverProxy is a subclass of EODelayedObserver that forwards change messages to a target object, allowing objects that don't inherit from EODelayedObserver to take advantage of this mechanism.

The major observer in Enterprise Objects Framework is EOEditingContext, which implements its **objectWillChange:** method to record a snapshot for the object about to change, register undo operations in an EOUndoManager, and record the changes needed to update objects in its EOObjectStore. Because some of these actions—such as examining the object's new state—can only be performed after the object has changed, an EOEditingContext sets up a delayed message to itself, which it gets at the end of the run loop. Observers that only need to examine an object after it has changed can use the delayed observer mechanism, described in the EODelayedObserver and EODelayedObserverQueue class specifications.

Registering an Observer

Objects that directly observe others must adopt the EOObserving protocol, which consists of the single method **objectWillChange:**. To register an object as an observer, invoke EOObserverCenter's **addObserver:forObject:** with the observer and the object to be observed. Once this is done, any time the object invokes its **willChange** method, the observer is sent an **objectWillChange:** message informing it of the pending change. You can also register an observer to be notified when any object changes using **addOmniscientObserver:**. This can be useful in certain situations, but as it's very costly to deal out

frequent change notifications, you should use omniscient observers sparingly. To unregister either kind of observer, simply use the corresponding **remove...** method.

Change Notification

Objects that are about to change invoke **willChange**, a method that the Framework adds to NSObject. This method invokes EOObserverCenter's **notifyObserversObjectWillChange:**, which sends an **objectWillChange:** message to all observers registered for the object that's changing, as well as to any omniscient observers. **notifyObserversObjectWillChange:** optimizes the process by suppressing redundant **objectWillChange:** messages when the same object invokes **willChange** several times in a row (as often happens when multiple properties are changed). Change notification is immediate, and takes place *before* the object's state changes. If you need to compare the object's state before and after the change, you must arrange to examine the new state at the end of the run loop.

You can suppress change notification when necessary, using the **suppressObserverNotification** and **enableObserverNotification** methods. While notification is suppressed, neither regular nor omniscient observers are informed of changes. These methods nest, so you can invoke **suppressObserverNotification** multiple times, and notification isn't reenabled until a matching number of **enableObserverNotification** message have been sent.

Method Types

Registering and unregistering observers	+ addObserver:forObject: + removeObserver:forObject: + addOmniscientObserver: + removeOmniscientObserver:
Notifying observers of change	+ notifyObserversObjectWillChange:
Getting observers	+ observersForObject: + observerForObject:ofClass:
Suppressing change notification	+ suppressObserverNotification + enableObserverNotification + observerNotificationSuppressCount

Class Methods

addObserver:forObject:

+ (void)**addObserver:**(id <EOObserving>)*anObserver* **forObject:**(id)*anObject*

Records *anObserver* to be notified with an **objectWillChange:** message when *anObject* changes.

See also: + **notifyObserversObjectWillChange:**, + **removeObserver:forObject:**, + **observersForObject:**

addOmniscientObserver:

+ (void)**addOmniscientObserver:**(id <EOObserving>)*anObserver*

Records *anObserver* to be notified with an **objectWillChange:** message when any object changes. This can cause significant performance degradation, and so should be used with care. The ominiscient observer must be prepared to receive the **objectWillChange:** message with a **nil** argument.

See also: + **notifyObserversObjectWillChange:**, + **addObserver:forObject:**, + **removeOmniscientObserver:**

enableObserverNotification

+ (void)**enableObserverNotification**

Counters a prior **suppressObserverNotification** message. When no such messages remain in effect, the **notifyObserversObjectWillChange:** method is reenabled. Raises an NSInternalInconsistencyException if not paired with a prior **suppressObserverNotification** message.

See also: + **observerNotificationSuppressCount**

notifyObserversObjectWillChange:

+ (void)**notifyObserversObjectWillChange:**(id)*anObject*

Unless change notification is suppressed, sends an **objectWillChange:** to all observers registered for *anObject* with that object as the argument, and sends that message to all omniscient observers as well. If invoked several times in a row with the same object, only the first invocation has any effect, since subsequent change notifications are redundant.

If an observer wants to ensure that it receives notification the next time the last object to change changes again, it should use the statement:

```
[EOObserverCenter notifyObserversObjectWillChange:nil];
```

See also: + **suppressObserverNotification**, + **addObserver:forObject:**, + **addOmniscientObserver:**

observerForObject:ofClass:

+ (id)**observerForObject:**(id)*anObject* **ofClass:**(Class)*aClass*

Returns an observer for *anObject* that's a kind of *aClass*. If more than one observer of *anObject* is a kind of *aClass*, the specific observer returned is undetermined. You can use **observersForObject:** instead to get all observers and examine their class membership.

observerNotificationSuppressCount

+ (unsigned int)**observerNotificationSuppressCount**

Returns the number of **suppressObserverNotification** messages in effect.

See also: + **suppressObserverNotification**, + **enableObserverNotification**

observersForObject:

+ (NSArray *)**observersForObject:**(id)*anObject*

Returns all observers of *anObject*.

See also: + **observerForObject:ofClass:**

removeObserver:forObject:

+ (void)**removeObserver:**(id <EOObserving>)*anObserver* **forObject:**(id)*anObject*

Removes *anObserver* as an observer of *anObject*.

See also: + **notifyObserversObjectWillChange:**, + **addObserver:forObject:**, + **observersForObject:**

removeOmniscientObserver:

+ (void)**removeOmniscientObserver:**(id <EOObserving>)*anObserver*

Unregisters *anObserver* as an observer of all objects. This can cause significant performance degradation, and so should be used with care.

See also: + **notifyObserversObjectWillChange:**, + **removeObserver:forObject:**, + **addOmniscientObserver:**

suppressObserverNotification

+ (void)**suppressObserverNotification**

Disables the **notifyObserversObjectWillChange:** method, so that no change notifications are sent. This method can be invoked multiple times; **enableObserverNotification** must then be invoked an equal number of times to reenable change notification.

See also: + **observerNotificationSuppressCount**