EODelayedObserverQueue

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

Class Description

EODelayedObserverQueue collects change notifications for observers of multiple objects and notifies them of the changes *en masse* during the application's run loop, according to their individual priorities. This style of notification is particularly useful for coalescing and prioritizing multiple changes; the Interface layer's EOAssociation classes use it extensively to update the user interface, for example. Instead of being told that an object will change, an EODelayedObserver is told that it did change, with a **subjectChanged** message, as described in the EODelayedObserver class specification. Delayed observation is thus not useful for comparing old and new states, but only for examining the new state.

The motivation for a delayed change notification mechanism arises mainly from issues in observing multiple objects. Any single change to an observed object typically requires the observer to update some state or perform an action. When many such objects change, it makes no sense to recalculate the new state and perform the action for each object. EODelayedObserverQueue allows these changes to be collected into a single notification. It further orders change notifications according to priorities, allowing observers to be updated in sequence according to dependencies among them. For example, an EOMasterDetailAssociation, which must update its detail EODisplayGroup according to the selection in the master *before* any redisplay occurs, has an earlier priority than the default for EOAssociations. This prevents regular EOAssociations from redisplaying old values and then displaying the new values after the EOMasterDetailAssociation updates.

Enqueuing a Delayed Observer

The **enqueueObserver:** method records an EODelayedObserver for later change notification. However, enqueuing is usually performed automatically by an EODelayedObserver in its **objectWillChange:** method. Hence, it's typically enough that an object being observed invoke **willChange** as needed. An EODisplayGroup, for example, does this (among many other things) on receiving an **objectsChangedInEditingContext:** message from its EOEditingContext.

Although you can create individual EODelayedObserverQueues using **alloc** and **init**, you typically use the single instance provided by the **defaultObserverQueue** class method. Using separate queues bypasses the prioritization mechanism (described below), which may cause problems between the objects using the separate queues. If you do use separate queues, your EODelayedObserver subclasses should record a

designated EODelayedObserverQueue that they always use, and implement **observerQueue** to return that object.

If you need to remove an enqueued observer, you can do so using the **dequeueObserver:** method. EODelayedObserver also defines the **discardPendingNotification** method, which removes the receiver from its designated queue. This is useful in an object's implementation of **dealloc**, for example, to prevent a change notification being sent to it.

Change Notification

The actual process of change notification is initiated by the **enqueueObserver:** messages that line observers up to receive notifications. Regardless of how many times **enqueueObserver:** is invoked for a particular observer, that observer is only put in the queue once. The first observer enqueued during the run loop also triggers the EODelayedObserverQueue to set up a delayed invocation of **notifyObserversUpToPriority:**, which causes it to receive that message at the end of the run loop. EODelayedObserver sets up this delayed invocation in NSDefaultRunLoopMode, but you can change that using **setRunLoopModes:**.

notifyObserversUpToPriority: cycles through the queue of EODelayedObservers in priority order, from EOObserverPriorityFirst to the priority given, sending each observer a **subjectChanged** message. Each time, it returns to the earliest priority (rather than continuing through the queue) in case the message resulted in another EODelayedObserver with a earlier priority being enqueued. This guarantees an optimal delivery of change notifications.

Observer Proxies

It may not always be possible for a custom observer class to inherit from EODelayedObserver. To aid such objects in participating in delayed change notifications, the Framework defines the EOObserverProxy class, which implements its **subjectChanged** method to invoke an action method of your custom object. You create an EOObserverProxy using the **initWithTarget:action:priority:** method, which records the "real" observer, the action method to invoke, and the priority at which the EOObserverProxy should be enqueued. Then, instead of registering the custom object as an observer of objects, you register the proxy (using EOObserverCenter's **addObserver:forObject:**). When it receives an **objectWillChange:** message, it enqueues itself for delayed change notification, receives the **subjectChanged** message from the EODelayedObserverQueue, and then sends the action message to the "real" observer.

Method Types

Creating instances	– init
Getting the default queue	+ defaultObserverQueue
Enqueuing and dequeuing observers	– enqueueObserver: – dequeueObserver:

Sending change notifications	– notifyObserversUpToPriority:
Configuring notification behavior	– setRunLoopModes: – runLoopModes

Class Methods

defaultObserverQueue

+ (EODelayedObserverQueue *)**defaultObserverQueue**

Returns the EODelayedObserverQueue that EODelayedObservers use by default.

Instance Methods

dequeueObserver:

– (void)**dequeueObserver:**(EODelayedObserver *)*anObserver*

Removes *anObserver* from the receiver.

See also: – **enqueueObserver:**

enqueueObserver:

– (void)**enqueueObserver:**(EODelayedObserver *)*anObserver*

Records *anObserver* to be sent a **subjectChanged** message. If *anObserver*'s priority is EOObserverPriorityImmediate it's immediately sent the message and not enqueued. Otherwise it's sent the message the next time **notifyObserversUpToPriority:** is invoked with a priority later than or equal to *anObserver*'s. Does nothing if *anObserver* is already recorded.

The first time this method is invoked during the run loop with an observer whose priority isn't EOObserverPriorityImmediate, it registers the receiver to be sent a **notifyObserversUpToPriority:** message at the end of the run loop, using EOFlushDelayedObserversRunLoopOrdering and the receiver's run loop modes. This causes enqueued observers up to a priority of EOObserverPrioritySixth to be notified automatically during each pass of the run loop.

This method does *not* retain *anObserver.* When *anObserver* is deallocated, it should invoke **dequeueObserver:** to remove itself from the queue.

See also: – **dequeueObserver:**, – **priority** (EODelayedObserver),
– **discardPendingNotification** (EODelayedObserver), – **runLoopModes**,
– **performSelector:target:argument:order:modes:** (NSRunLoop class of the Foundation Kit)

init

– (id)**init**

Initializes a newly allocated EODelayedObserverQueue with NSDefaultRunLoopMode as its only run loop mode. This is the designated initializer for the EODelayedObserverQueue class. Returns **self**.

notifyObserversUpToPriority:

– (void)**notifyObserversUpToPriority:**(EOObserverPriority)*priority*

Sends **subjectChanged** messages to all of the receiver's enqueued observers whose priority is *priority* or earlier. This method cycles through the receiver's enqueued observers in priority order, sending each a **subjectChanged** message and then returning to the very beginning of the queue, in case another observer with an earlier priority was enqueued as a result of the message.

EODelayedObserverQueue invokes this method automatically as needed during the run loop, with a *priority* of EOObserverPrioritySixth.

See also: – **enqueueObserver:**, – **priority** (EODelayedObserver)

runLoopModes

– (NSArray *)**runLoopModes**

Returns the receiver's run loop modes.

See also: – **setRunLoopModes:**

setRunLoopModes:

– (void)**setRunLoopModes:**(NSArray *)*modes*

Sets the receiver's run loop modes to *modes*.

See also: – **runLoopModes**