NSNotificationQueue

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: Foundation/NSNotificationQueue.h

Class Description

NSNotificationQueue objects (or simply, *notification queues*) act as buffers for notification centers (instances of NSNotificationCenter). A notification queue maintains notifications (instances of NSNotification) generally in a First In First Out (FIFO) order. When a notification rises to the front of the queue, the queue posts it to the notification center, which in turn dispatches the notification to all objects registered as observers.

Every thread has a default notification queue, which is associated with the default notification center for the task. You can create your own notification queues and have multiple queues per center and thread.

Coalescing Notifications

NSNotificationQueue contributes two important features to the Foundation Kit's notification mechanism: the coalescing of notifications and asynchronous posting. *Coalescing* is a process that removes notifications in the queue that are similar to the notification just queued. If the new item is similar to a notification already queued, the new one isn't queued, and all similar notifications (except the first one in the queue) are removed. However, you should not depend on this particular coalescing behavior.

You indicate the criteria for similarity by specifying NSNotificationCoalescing constants in the third argument of **enqueueNotification:postingStyle:coalesceMask:forModes:**

- **NSNotificationNoCoalescing**. Do not coalesce notifications in the queue.
- **NSNotificationCoalescingOnName**. Coalesce notifications with the same name.
- **NSNotificationCoalescingOnSender**. Coalesce notifications with the same object.

You can OR the constants together to specify more than one.

Asynchronously Posting Notifications

With NSNotificationCenter's **postNotification:** and its variants, you can post a notification immediately to a notification center. However, the invocation of the method is synchronous: Before the posting object can resume its thread of execution, it must wait until the notification center dispatches the notification to all

observers and returns. With NSNotificationQueue's **enqueueNotification:postingStyle:** and **enqueueNotification:postingStyle:coalesceMask:forModes**:, however, you can post a notification asynchronously by putting it on the queue. These methods immediately return to the invoking object after putting the notification in the queue.

Posting to a notification queue can occur in one of three different styles. The posting style is an argument to both **enqueueNotification:...** methods:

- **NSPostASAP**. The notification is posted at the end of the current notification callout or timer.
- **NSPostWhenIdle**. The notification is posted when the run loop is idle.
- **NSPostNow**. The notification is posted immediately after coalescing.

Note: See "Enqueuing with Different Posting Styles," below, for details on and examples of enqueuing notifications with the three **postingStyle:** constants.

What is the difference between enqueuing notifications with **NSPostNow** and posting notifications (using NSNotificationCenter's **postNotification...** methods)? Both post notifications immediately (but synchronously) to the notification center. The difference is that **enqueueNotification:...** (with **NSPostNow** as the posting style) coalesces notifications in the queue before posting while **postNotification:** does not.

enqueueNotification:postingStyle:coalesceMask:forModes: also allows you to control the posting of a notification based on the run loop mode. For example, if you specify NSModalPanelRunLoopMode, the notification will not be posted unless the current run loop is in NSModalPanelRunLoopMode. See the NSRunLoop class specification for more information on run loop modes.

Enqueuing with Different Posting Styles

Any notification queued with the **NSPostASAP** style is posted to the notification center when the code executing in the current run loop callout completes. Callouts can be Application Kit event messages, file descriptor changes, timers, or other asynchronous notifications. You typically use the **NSPostASAP** posting style for an expensive resource, such as the Display PostScript server. When many clients draw on the window buffer during a callout, it's expensive to flush the buffer to the Display PostScript server after every draw operation. So in this case, each **draw...** method enqueues some notification such as "FlushTheServer" with coalescing on name and object specified, and a posting style of **NSPostASAP**. As a result, only one of those notifications is dispatched at the end of the current callout, and the window buffer is flushed only once.

A notification queued with the **NSPostWhenIdle** style is posted only when the run loop is in a wait state. In this state, there's nothing in the run loop's input channels, be it timers or other asynchronous notifications. A typical example of queuing with the **NSPostWhenIdle** style occurs when the user types text, and the program displays the size of the text in bytes somewhere. It would be very expensive (and not very useful) to update the text field size after each character the user types, especially if the user types quickly. In this case, the program queues a notification after each character typed such as "ChangeTheDisplayedSize" with coalescing turned on and a posting style of **NSPostWhenIdle**. When the user stops typing, the single "ChangeTheDisplayedSize" notification in the queue (due to coalescing) is posted when the run loop is in a wait state, and the display is updated.

A notification queued with **NSPostNow** is posted immediately after coalescing to the notification center. You queue a notification with **NSPostNow** (or post one with NSNotificationCenter's **postNotification:**) when you do not require asynchronous calling behavior. For many programming situations, synchronous behavior is not only allowable but desirable: You want the notification center to return after dispatching so you can be sure that observing objects have received the notification. Of course, you should use **enqueueNotification...** with **NSPostNow** rather than use **postNotification:** when there are similar notifications in the queue that you want to remove through coalescing.

Method Types

Creating and Initializing Notification Queues

+ defaultQueue
– init
– initWithNotificationCenter:

Inserting and Removing Notifications From a Queue

– dequeueNotificationsMatching:coalesceMask:
– enqueueNotification:postingStyle:
– enqueueNotification:postingStyle:coalesceMask:
forModes:

Class Methods

defaultQueue

+ (NSNotificationQueue *)**defaultQueue**

Returns the default NSNotificationQueue object for the current thread. This object always uses the default notification center object for the same task.

Instance Methods

dequeueNotificationsMatching:coalesceMask:

– (void)**dequeueNotificationsMatching:**(NSNotification *)*notification*
coalesceMask:(unsigned int)*coalesceMask*

Removes all notifications from the queue that match *notification*'s attributes as specified by *coalesceMask*. The mask (set using NSNotificationCoalescing constants) can specify notification name, notification object, or both.

enqueueNotification:postingStyle:

– (void)**enqueueNotification:**(NSNotification *)*notification*
postingStyle:(NSPostingStyle)*postingStyle*

Puts *notification* in the queue. The queue posts *notification* to the notification center at the time indicated by *postingStyle.* The notification queue posts in all run loop modes, and it coalesces only notifications in the queue that match both the notification's name and object.

This method invokes **enqueueNotification:postingStyle:coalesceMask:forModes:**.

enqueueNotification:postingStyle:coalesceMask:forModes:

– (void)**enqueueNotification:**(NSNotification *)*notification*
postingStyle:(NSPostingStyle)*postingStyle*
coalesceMask:(unsigned int)*coalesceMask*
forModes:(NSArray *)*modes*

Puts *notification* in the queue. The queue posts *notification* to the notification center at the time indicated by *postingStyle,* but only if the run loop is in a mode identified by one of the string objects in the *modes* array. The notification queue coalesces related notifications in the queue as specified by *coalesceMask* (set using NSNotificationCoalescing constants). If *modes* is **nil**, all run loop modes are valid for posting.

init

– (id)**init**

Initializes and returns an NSNotificationQueue that uses the default notification center to post notifications. This method invokes **initWithNotificationCenter:** with the default notification center as the argument.

initWithNotificationCenter:

– (id)**initWithNotificationCenter:**(NSNotificationCenter *)*notificationCenter*

Initializes and returns an NSNotificationQueue that uses the notification center specified in *notificationCenter* to post notifications. This method is the designated initializer.