NSRunLoop

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation/NSRunLoop.h

lass Description

The NSRunLoop class declares the programmatic interface to objects that manage input sources. An NSRunLoop processes input for sources such as mouse and keyboard events from the window system, NSPorts, NSTimers, and NSConnections.

In general, your application won't need to either create or explicitly manage NSRunLoop objects. Each thread has an NSRunLoop object automatically created for it. Each process begins with a default thread and therefore has a default run loop.

If you do want to perform your own explicit run loop management, you do so by accessing the current thread's run loop (returned by the class method **currentRunLoop**). You must specify two things: the *input sources*, which are the objects from which the run loop will receive information, and an *input mode*, which specifies the type of input to be received. The currently defined input modes are:

NSDefaultRunLoopMode Use this mode to deal with input sources other than NSConnections. Defined in the **Foundation/NSRunLoop.h** header file. This is the most commonly used run loop mode.

NSConnectionReplyMode Use this mode to indicate NSConnection objects waiting for replies. Defined in the **Foundation/NSConnection.h** header file. You rarely need to use this mode.

In addition, the Application Kit defines these modes:

NSModalPanelRunLoopModeUse this mode when waiting for input from a modal panel, such as NSSavePanel or NSOpenPanel. Defined in the **AppKit/NSApplication.h** header file.

NSEventTrackingRunLoopModeUse this mode for event tracking loops. Defined in the **AppKit/NSApplication.h** header file.

You associate a list of input sources with each input mode. There are two general types of input sources to a run loop: asynchronous (input arrives at unpredictable intervals) and synchronous (input arrives at regular intervals). NSPort objects represent asynchronous input sources, and NSTimer objects represent synchronous input sources. Each input source has a limit date associated with it. For NSPorts, the limit date is a timeout value, after which input from that port is no longer timely. For NSTimers, the limit date specifies when the timer should fire. (When a timer fires, it sends a specified message to a specified object, and it may be scheduled to fire again later. See the NSTimer class specification for more information.)

When an NSRunLoop runs, it polls each of the sources for the input mode to determine which one has the earliest limit date. During this polling, the input sources may process any input they have queued. Once the NSRunLoop determines the earliest limit date for this input mode, it waits for input from the operating system until that limit date. If input arrives, it is processed. At that point, the NSRunLoop may either return or it may continue, depending on which method was used to run the loop.

For example:

```
NSRunLoop *theLoop = [NSRunLoop currentRunLoop];

[theLoop acceptInputForMode:NSDefaultRunLoopMode beforeDate:[theLoop
    limitDateForMode:NSDefaultRunLoopMode]];
```

The method **limitDateForMode:** returns the earliest limit date of all the input sources for the mode NSDefaultRunLoopMode. **acceptInputForMode:beforeDate:** runs the loop until that date, processing any input it receives until that time. As a convenience, you can use **runMode:beforeDate:** instead. It invokes **acceptInputForMode:beforeDate:** and **limitDateForMode:** with the mode you supply.

To continuously run in NSDefaultRunLoopMode, you can use either of the methods **run** or **runUntilDate:**. To run another mode continuously, invoke **runMode:beforeDate:** in a loop with a date far in the future:

```
while ([[NSRunLoop currentRunLoop] runMode:NSModalPanelRunLoopMode
    beforeDate:[NSDate distantFuture]])
    ;
```

Note: Regardless of the date you specify in **runMode:beforeDate:**, a run loop with nothing to do (that is no sources from which to receive input) will exit immediately. You must add the input sources to the run loop mode before you start the run loop.

Method Types

Accessing the current run loop	+ currentRunLoop – currentMode – limitDateForMode:
Adding timers	– addTimer:forMode:
Managing ports	– addPort: forMode: – removePort:forMode:
Setting up for server processes	– configureAsServer
Running a loop	– run – runUntilDate: – runMode:beforeDate: – acceptInputForMode:beforeDate:
Sending messages	– performSelector:target:argument:order:modes: – cancelPerformSelector:target:argument:

lass Methods

currentRunLoop

+ (NSRunLoop *)**currentRunLoop**

Returns the NSRunLoop for the current thread.

See also: **– currentMode**

nstance Methods

acceptInputForMode:beforeDate:

– (void)**acceptInputForMode:**(NSString *)*mode* **beforeDate:**(NSDate *)*limitDate*

Blocks awaiting input from the ports in the port list for the input mode *mode* until the time specified by *limitDate*. Use the **limitDateForMode:** method to calculate *limitDate*. If input arrives, it is processed using the NSPort delegates. This method does not check the timers associated with *mode*, thus it does not fire timers even if their scheduled fire dates have passed.

addPort: forMode:

– (void)**addPort:**(NSPort *)*aPort* **forMode:**(NSString *)*mode*

Adds *aPort* to be monitored by the receiver in the input mode *mode*. The receiver maintains a count of the number of ports added, and the same number must be removed.

See also: **– removePort:forMode:**

addTimer:forMode:

– (void)**addTimer:**(NSTimer *)*aTimer*
    forMode:(NSString *)*mode*

Registers the timer *aTimer* with input mode *mode*. The run loop causes the timer to fire on or after its scheduled fire date. Timers have an Objective-C message associated with them. When a timer fires, it sends its message to the appropriate object. To remove a timer from a mode, send the **invalidate** message to the timer.

cancelPerformSelector:target:argument:

– (void)**cancelPerformSelector:**(SEL)*aSelector* **target:**(id)*target* **argument:**(id)*anArgument*

Cancels the sending of a message previously scheduled using **performSelector:target:argument:order:modes:**. The *aSelector* message with argument *anArgument* will not be sent to *target*.

configureAsServer

- (void)**configureAsServer**

Performs all necessary configuration to make the run loop suitable for use by a server process. For example, on the Microsoft Windows platform, this method adds a port that receives messages from the WIN32 event queue. This enables the run loop to notice when the user logs out, allowing the server process to exit gracefully.

currentMode

– (NSString *)**currentMode**

Returns the current input mode. The current mode is set by **limitDateForMode:** and **acceptInputForMode:beforeDate:**.

See also: **+ currentRunLoop**

limitDateForMode:

– (NSDate *)**limitDateForMode:**(NSString *)*mode*

Polls *mode*'s input sources for their limit date (if any) and returns the earliest limit date for this mode. Uses the NSPort delegate method **limitDateForMode:** to determine the limit dates of ports. Fires timers if their limit dates have passed. Polls ports for activities appropriate for *mode*. Returns **nil** if there are no input sources for this mode.

performSelector:target:argument:order:modes:

– (void)**performSelector:**(SEL)*aSelector* **target:**(id)*target* **argument:**(id)*anArgument*
order:(unsigned)*order* **modes:**(NSArray *)*modes*

Schedules the sending of an *aSelector* message. The *aSelector* message will be sent to *target* with argument *anArgument* after the run loop has completed an iteration in any of the input modes specified in *modes*. *order* assigns a priority to the messages. If multiple messages are scheduled to be sent, the messages with a lower *order* value are sent before messages with a higher *order* value.

This method returns before the *aSelector* message is sent. The *aSelector* method should not have a significant return value and should take a single argument of type **id**. The NSRunLoop does not retain the *target* and *anArgument* objects.

Use this method is you want multiple messages to be sent after the current event has been processed and you want to make sure that these messages are sent in a certain order.

See also: – **cancelPerformSelector:target:argument:**

removePort:forMode:

– (void)**removePort:**(NSPort *)*aPort*
  forMode:(NSString *)*mode*

Removes *aPort* from the list of ports being monitored by the receiver in input mode *mode*. The receiver maintains a count of the ports added, and the same number of ports must be removed. Ports are automatically removed from modes if they are detected to be invalid.

See also: **– addPort:forMode:**

run

– (void)**run**

Runs the loop in NSDefaultRunLoopMode by repeatedly invoking **runMode:beforeDate:** until the limit dates for all of the input sources have passed.

See also: – **runUntilDate:**

runMode:beforeDate:

– (BOOL)**runMode:**(NSString *)*mode*
  beforeDate:(NSDate *)*limitDate*

Runs the loop once by invoking **acceptInputForMode:beforeDate:**, accepting input for mode *mode* until a limit date. The limit date is determined by using the earliest of *limitDate* and the limit dates set for all input sources in this mode. Returns NO without starting the run loop if the limit dates for all of *mode*'s input sources have passed; otherwise returns YES.

runUntilDate:

– (void)**runUntilDate:**(NSDate *)*limitDate*

Runs the loop in NSDefaultRunLoopMode by repeatedly invoking **runMode:beforeDate:** until *limitDate* or until the limit dates for all of the input sources have passed.

See also: – **run**