Run Loops

(Legacy)

Cocoa > Events & Other Input

2008-10-15

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction to Run Loops

Important: The information in this document is superseded by the information in *Threading Programming Guide*. For information about how to configure a run loop for your custom threads, see that document instead.

Organization of This Document

This programming topic contains the follow articles:

- "Run Loops" (page 7) provides an overview of what a run loop is and how it operates.
- "Input Modes" (page 9) describes how input modes are used by run loops to categorize input sources.
- "Getting the Run Loop" (page 11) describes how to obtain the current run loop object.
- "Adding Input Sources" (page 13) describes how to add input sources to a run loop.
- "Running the Run Loop" (page 15) describes the ways you enter the run loop.

Run Loops

Event-driven applications receive their events in a run loop. A run loop monitors sources of input to the application and dispatches control when sources become ready for processing. When processing is complete, control passes back to the run loop which then waits for the next event. Possible events include mouse and keyboard events from the window system, the arrival of data on ports, the firing of timers, and distributed object requests.

The NSRunLoop class declares the programmatic interface to objects that manage input sources, the objects from which the run loop receives information. 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.

Besides managing input sources, NSRunLoop also provides support for delayed actions that are event-driven rather than timer driven. NSWindow uses delayed actions to perform screen updates on dirty views. NSNotificationQueue uses them to post notifications queued with the modes `NSPostASAP` and `NSPostWhenIdle`. You can request a delayed action with the NSRunLoop method `performSelector:target:argument:order:modes:`; the indicated method is then sent to the target at the start of the next run loop.

In general, your application does not need to either create or explicitly manage NSRunLoop objects. Each NSThread, including the application's main thread, has an NSRunLoop object automatically created for it. However, only the main thread in an application using the Application Kit automatically runs its run loop; additional threads (or Foundation Kit tools) have to explicitly run the run loop themselves. To access the current thread's default run loop, use the NSRunLoop class method `currentRunLoop`.

When an NSRunLoop runs, it polls each of the sources for the input mode to determine if any need processing. Only one is processed per loop. If no input sources are processed, NSRunLoop waits for input from the operating system. The run loop waits until input arrives or a timeout—provided when starting the run loop—is exceeded. At this point, the NSRunLoop may either return or it may continue, depending on which method was used to run the loop.

Input Modes

Run loops can be run in different modes. A mode, which is identified by an arbitrary string, defines a collection of input sources that is monitored while the run loop is in that mode. For example, you can have one mode that runs while the application is idle, waiting for all types of events to process, and another that only listens to a particular port, waiting for a response from a distributed object request. You do not want to handle keyboard events in the latter case, since the application has not finished processing an earlier event which caused the distributed object request to be made.

NSRunLoop defines this input mode:

Input mode	Description
`NSDefaultRunLoopMode`	Use this mode to deal with input sources other than NSConnections. This is the most commonly used run loop mode.

In addition, NSConnection defines this mode:

Input mode	Description
`NSConnectionReplyMode`	Use this mode to indicate NSConnection objects waiting for replies. You rarely need to use this mode.

And NSApplication defines these modes:

Input mode	Description
`NSModalPanelRunLoopMode`	Use this mode when waiting for input from a modal panel, such as NSSavePanel or NSOpenPanel.
`NSEventTrackingRunLoopMode`	Use this mode for event tracking loops.

You associate a list of input sources with each input mode. Sources are added with either the NSRunLoop methods `addPort:forMode:` or `addTimer:forMode:` or one of the convenience methods provided by NSConnection, NSPort, and NSTimer. Input sources can be added to multiple input modes.

You create additional modes by specifying a new mode name when adding an input source to that mode.

Getting the Run Loop

When using an application built using the Application Kit, a run loop is created and run automatically. If you need to access this run loop, use the NSRunLoop class method `currentRunLoop`.

Additional run loops are created for each additional NSThread and also can be accessed by invoking `currentRunLoop` from each thread. These run loops do not have any input sources and are not running when the thread begins. You must add input sources to them and start the run loop yourself.

Warning: The NSRunLoop class is generally not considered to be thread-safe and its methods should only be called within the context of the current thread. You should never try to call the methods of an NSRunLoop object running in a different thread, as doing so might cause unexpected results.

Adding Input Sources

In most cases, input source objects add themselves to the current run loop as needed, but you can add them manually to get greater control over their behavior.

The NSTimer class method `scheduledTimerWithTimeInterval:invocation:repeats:`, for example, creates a new timer object and adds it to the `NSDefaultRunLoopMode` mode of the current run loop. If you instead create the timer with `timerWithTimeInterval:invocation:repeats:`, you must add it manually to the run loop with the NSRunLoop instance method `addTimer:forMode:`, which allows you to specify a different mode.

NSPort objects are usually used as part of an NSConnection, which automatically adds its receive port to the appropriate modes as needed. If you have a stand-alone port object, you can manually add it to the run loop with the NSRunLoop method `addPort:forMode:`.

Running the Run Loop

You have numerous ways in which to run the run loop. Using `run`, control is passed to the run loop until all input sources in the `NSDefaultRunLoopMode` mode have been removed; if there are no input sources, the run loop returns immediately:

```
[[NSRunLoop currentRunLoop] run];
```

To specify a time at which the run loop should stop processing events and return control, use `runUntilDate:`:

```
[[NSRunLoop currentRunLoop] runUntilDate:aDate];
```

To specify a mode other than `NSDefaultRunLoopMode`, use `runMode:beforeDate:`. This method runs the run loop only once; it returns either after it processes a single input source or the *`beforeDate`* date is reached. To run any mode continuously, invoke `runMode:beforeDate:` in a loop with a date far in the future:

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

The return value of `runMode:beforeDate:` indicates whether the run loop is still running; if the run loop is empty (in other words, it has no input sources) `runMode:beforeDate:` returns `NO` and the `while` loop exits.

Finally, to conditionalize the run loop so that you can define an exit condition, include a test in the loop surrounding the `runMode:beforeDate:` invocation:

```
double resolution = 1.0;
BOOL endRunLoop = NO;
BOOL isRunning;
do {
    NSDate* next = [NSDate dateWithTimeIntervalSinceNow:resolution];
    isRunning = [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode
                beforeDate:next];
} while (isRunning && !endRunLoop);
```

In this snippet, the `endRunLoop` variable is the test condition indicating when to break out of the run loop. It may be either a global variable or an instance variable that is set to `YES` from a run loop callout when it is time to exit the run loop.

Note: Regardless of the date you specify in `runMode:beforeDate:` and `runUntilDate:`, a run loop with nothing to do (that is no sources from which to receive input) exits immediately. You must add the input sources to the run loop mode before you start the run loop. Other parts of the system may add their own sources to a particular run loop mode, but do not depend on this always being the case. Add an empty NSPort to a run loop if you need to guarantee that it does not exit immediately

Document Revision History

This table describes the changes to *Run Loops*.

Date	Notes
2008-10-15	This document is now replaced by the Threading Programming Guide.
2005-01-11	Added a warning about NSRunLoop thread safety. Fixed minor bugs.
2002-11-12	Corrected error in sample code in "Running the Run Loop" (page 15).
	Revision history was added to existing topic.