CFRunLoop Reference

Data Management: Event Handling

2009-05-16

Contents

CFRunLoop Reference 5

Overview 5
Functions by Task 6
    Getting a Run Loop 6
    Starting and Stopping a Run Loop 6
    Managing Sources 6
    Managing Observers 7
    Managing Run Loop Modes 7
    Managing Timers 7
    Scheduling Blocks 7
    Getting the CFRunLoop Type ID 7
Functions 8
    CFRunLoopAddCommonMode 8
    CFRunLoopAddObserver 8
    CFRunLoopAddSource 9
    CFRunLoopAddTimer 10
    CFRunLoopContainsObserver 11
    CFRunLoopContainsSource 11
    CFRunLoopContainsTimer 12
    CFRunLoopCopyAllModes 13
    CFRunLoopCopyCurrentMode 13
    CFRunLoopGetCurrent 14
    CFRunLoopGetMain 14
    CFRunLoopGetNextTimerFireDate 15
    CFRunLoopGetTypeID 15
    CFRunLoopIsWaiting 16
    CFRunLoopPerformBlock 16
    CFRunLoopRemoveObserver 17
    CFRunLoopRemoveSource 17
    CFRunLoopRemoveTimer 18
    CFRunLoopRun 19
    CFRunLoopRunInMode 19
    CFRunLoopStop 20
    CFRunLoopWakeUp 21
Data Types 22
    CFRunLoopRef 22
Constants 22
    CFRunLoopRunInMode Exit Codes 22
    Common Mode Flag 23
    Default Run Loop Mode 23

Document Revision History 25

CFRunLoop Reference

Derived From:	*CFType Reference*
Framework:	CoreFoundation/CoreFoundation.h
Companion guide	Run Loops
Declared in	CFRunLoop.h

Overview

A CFRunLoop object monitors sources of input to a task and dispatches control when they become ready for processing. Examples of input sources might include user input devices, network connections, periodic or time-delayed events, and asynchronous callbacks.

Three types of objects can be monitored by a run loop: sources (*CFRunLoopSource Reference*), timers (*CFRunLoopTimer Reference*), and observers (*CFRunLoopObserver Reference*). To receive callbacks when these objects need processing, you must first place these objects into a run loop with `CFRunLoopAddSource` (page 9), `CFRunLoopAddTimer` (page 10), or `CFRunLoopAddObserver` (page 8). You can later remove an object from the run loop (or invalidate it) to stop receiving its callback.

Run loops have different modes in which they can run. Each mode has its own set of objects that the run loop monitors while running in that mode. Core Foundation defines a default mode, `kCFRunLoopDefaultMode` (page 23), to hold objects that should be monitored while the application (or thread) is sitting idle. Additional modes are created automatically when an object is added to an unrecognized mode. Each run loop has its own independent set of modes.

Core Foundation also defines a special pseudo-mode `kCFRunLoopCommonModes` (page 23) to hold objects that should be shared by a set of "common" modes. A mode is added to the set of "common" modes by calling `CFRunLoopAddCommonMode` (page 8). The default mode, `kCFRunLoopDefaultMode` (page 23), is always a member of the set of common modes. The `kCFRunLoopCommonModes` (page 23) constant is never passed to `CFRunLoopRunInMode` (page 19). Each run loop has its own independent set of common modes.

There is exactly one run loop per thread. You neither create nor destroy a thread's run loop. Core Foundation automatically creates it for you as needed. You obtain the current thread's run loop with `CFRunLoopGetMain` (page 14). Call `CFRunLoopRun` (page 19) to run the current thread's run loop in the default mode until the run loop is stopped with `CFRunLoopStop` (page 20). You can also call `CFRunLoopRunInMode` (page 19) to run the current thread's run loop in a specified mode for a set period of time (or until the run loop is stopped). A run loop can only run if the requested mode has at least one source or timer to monitor.

Run loops can be run recursively. You can call `CFRunLoopRun` (page 19) or `CFRunLoopRunInMode` (page 19) from within any run loop callout and create nested run loop activations on the current thread's call stack. You are not restricted in which modes you can run from within a callout. You can create another run loop activation running in any available run loop mode, including any modes already running higher in the call stack.

Cocoa and Carbon each build upon CFRunLoop to implement their own higher-level event loop. When writing a Cocoa or Carbon application, you can add your sources, timers, and observers to their run loop objects and modes. Your objects will then get monitored as part of the regular application event loop. Use the NSRunLoop instance method `getCFRunLoop` to obtain the CFRunLoop corresponding to a Cocoa run loop. In Carbon applications, use the `GetCFRunLoopFromEventLoop` function.

Functions by Task

Getting a Run Loop

`CFRunLoopGetCurrent` (page 14)
Returns the CFRunLoop object for the current thread.

`CFRunLoopGetMain` (page 14)
Returns the main CFRunLoop object.

Starting and Stopping a Run Loop

`CFRunLoopRun` (page 19)
Runs the current thread's CFRunLoop object in its default mode indefinitely.

`CFRunLoopRunInMode` (page 19)
Runs the current thread's CFRunLoop object in a particular mode.

`CFRunLoopWakeUp` (page 21)
Wakes a waiting CFRunLoop object.

`CFRunLoopStop` (page 20)
Forces a CFRunLoop object to stop running.

`CFRunLoopIsWaiting` (page 16)
Returns a Boolean value that indicates whether the run loop is waiting for an event.

Managing Sources

`CFRunLoopAddSource` (page 9)
Adds a CFRunLoopSource object to a run loop mode.

`CFRunLoopContainsSource` (page 11)
Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopSource object.

`CFRunLoopRemoveSource` (page 17)
Removes a CFRunLoopSource object from a run loop mode.

Managing Observers

`CFRunLoopAddObserver` (page 8)
: Adds a CFRunLoopObserver object to a run loop mode.

`CFRunLoopContainsObserver` (page 11)
: Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopObserver object.

`CFRunLoopRemoveObserver` (page 17)
: Removes a CFRunLoopObserver object from a run loop mode.

Managing Run Loop Modes

`CFRunLoopAddCommonMode` (page 8)
: Adds a mode to the set of run loop common modes.

`CFRunLoopCopyAllModes` (page 13)
: Returns an array that contains all the defined modes for a CFRunLoop object.

`CFRunLoopCopyCurrentMode` (page 13)
: Returns the name of the mode in which a given run loop is currently running.

Managing Timers

`CFRunLoopAddTimer` (page 10)
: Adds a CFRunLoopTimer object to a run loop mode.

`CFRunLoopGetNextTimerFireDate` (page 15)
: Returns the time at which the next timer will fire.

`CFRunLoopRemoveTimer` (page 18)
: Removes a CFRunLoopTimer object from a run loop mode.

`CFRunLoopContainsTimer` (page 12)
: Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopTimer object.

Scheduling Blocks

`CFRunLoopPerformBlock` (page 16)
: Enqueues a Block on a given runloop to be executed as the runloop cycles in specified modes.

Getting the CFRunLoop Type ID

`CFRunLoopGetTypeID` (page 15)
: Returns the type identifier for the CFRunLoop opaque type.

Functions

CFRunLoopAddCommonMode

Adds a mode to the set of run loop common modes.

```
void CFRunLoopAddCommonMode (
   CFRunLoopRef rl,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify. Each run loop has its own independent list of modes that are in the set of common modes.

mode

The run loop mode to add to the set of common modes of *rl*.

Discussion

Sources, timers, and observers get registered to one or more run loop modes and only run when the run loop is running in one of those modes. Common modes are a set of run loop modes for which you can define a set of sources, timers, and observers that are shared by these modes. Instead of registering a source, for example, to each specific run loop mode, you can register it once to the run loop's common pseudo-mode and it will be automatically registered in each run loop mode in the common mode set. Likewise, when a mode is added to the set of common modes, any sources, timers, or observers already registered to the common pseudo-mode are added to the newly added common mode.

Once a mode is added to the set of common modes, it cannot be removed.

The Add, Contains, and Remove functions for sources, timers, and observers operate on a run loop's set of common modes when you use the constant `kCFRunLoopCommonModes` (page 23) for the run loop mode.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopCopyAllModes` (page 13)

`CFRunLoopCopyCurrentMode` (page 13)

Declared In

`CFRunLoop.h`

CFRunLoopAddObserver

Adds a CFRunLoopObserver object to a run loop mode.

```
void CFRunLoopAddObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

rl
: The run loop to modify.

observer
: The run loop observer to add.

mode
: The run loop mode to which to add *observer*. Use the constant `kCFRunLoopCommonModes` (page 23) to add *observer* to the set of objects monitored by all the common modes.

Discussion

A run loop observer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

If *rl* already contains *observer* in *mode*, this function does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopContainsObserver` (page 11)

`CFRunLoopRemoveObserver` (page 17)

Declared In

`CFRunLoop.h`

CFRunLoopAddSource

Adds a CFRunLoopSource object to a run loop mode.

```
void CFRunLoopAddSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl
: The run loop to modify.

source
: The run loop source to add.

mode
: The run loop mode to which to add *source*. Use the constant `kCFRunLoopCommonModes` (page 23) to add *source* to the set of objects monitored by all the common modes.

Discussion

If *source* is a version 0 source, this function calls the `schedule` callback function specified in the context structure for *source*. See `CFRunLoopSourceContext` for more details.

A run loop source can be registered in multiple run loops and run loop modes at the same time. When the source is signaled, whichever run loop that happens to detect the signal first will fire the source.

If *rl* already contains *source* in *mode*, this function does nothing.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFRunLoopContainsSource` (page 11)
`CFRunLoopRemoveSource` (page 17)

Related Sample Code
audioburntest
bulkerase
CFLocalServer
databurntest
ImageClient

Declared In
`CFRunLoop.h`

CFRunLoopAddTimer

Adds a CFRunLoopTimer object to a run loop mode.

```
void CFRunLoopAddTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl
The run loop to modify.

timer
The run loop timer to add.

mode
The run loop mode of *rl* to which to add *timer*. Use the constant `kCFRunLoopCommonModes` (page 23) to add *timer* to the set of objects monitored by all the common modes.

Discussion
A run loop timer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

If *rl* already contains *timer* in *mode*, this function does nothing.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GLUT
SampleUSBMIDIDriver

Worm

Declared In
`CFRunLoop.h`

CFRunLoopContainsObserver

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopObserver object.

```
Boolean CFRunLoopContainsObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

`rl`
The run loop to examine.

`observer`
The run loop observer for which to search.

`mode`
The run loop mode in which to search for *`observer`*. Use the constant `kCFRunLoopCommonModes` (page 23) to search for *`observer`* in the set of objects monitored by all the common modes.

Return Value
`true` if *`observer`* is in mode *`mode`* of the run loop *`rl`*, otherwise `false`.

Discussion
If *`observer`* was added to `kCFRunLoopCommonModes` (page 23), this function returns `true` if *`mode`* is either `kCFRunLoopCommonModes` (page 23) or any of the modes that has been added to the set of common modes.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFRunLoopAddObserver` (page 8)
`CFRunLoopRemoveObserver` (page 17)

Declared In
`CFRunLoop.h`

CFRunLoopContainsSource

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopSource object.

```
Boolean CFRunLoopContainsSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl
: The run loop to examine.

source
: The run loop source for which to search.

mode
: The run loop mode of *rl* in which to search. Use the constant `kCFRunLoopCommonModes` (page 23) to search for *source* in the set of objects monitored by all the common modes.

Return Value

`true` if *source* is in mode *mode* of the run loop *rl*, otherwise `false`.

Discussion

If *source* was added to `kCFRunLoopCommonModes` (page 23), this function returns `true` if *mode* is either `kCFRunLoopCommonModes` (page 23) or any of the modes that has been added to the set of common modes.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddSource` (page 9)

`CFRunLoopRemoveSource` (page 17)

Related Sample Code

HID Manager Basics

HID Utilities Source

SampleUSBMIDIDriver

Declared In

`CFRunLoop.h`

CFRunLoopContainsTimer

Returns a Boolean value that indicates whether a run loop mode contains a particular CFRunLoopTimer object.

```
Boolean CFRunLoopContainsTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl
: The run loop to examine.

timer
: The run loop timer for which to search.

mode

The run loop mode of *rl* in which to search for *timer*. Use the constant `kCFRunLoopCommonModes` (page 23) to search for *timer* in the set of objects monitored by all the common modes.

Return Value

`true` if *timer* is in mode *mode* of the run loop *rl*, `false` otherwise.

Discussion

If *timer* was added to `kCFRunLoopCommonModes` (page 23), this function returns `true` if *mode* is either `kCFRunLoopCommonModes` (page 23) or any of the modes that has been added to the set of common modes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

CFRunLoopCopyAllModes

Returns an array that contains all the defined modes for a CFRunLoop object.

```
CFArrayRef CFRunLoopCopyAllModes (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value

An array that contains all the run loop modes defined for *rl*. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddCommonMode` (page 8)

`CFRunLoopCopyCurrentMode` (page 13)

Declared In

`CFRunLoop.h`

CFRunLoopCopyCurrentMode

Returns the name of the mode in which a given run loop is currently running.

```
CFStringRef CFRunLoopCopyCurrentMode (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value
The mode in which `rl` is currently running; `NULL` if `rl` is not running. Ownership follows the Create Rule.

Discussion
When run on the current thread's run loop, the returned value identifies the run loop mode that made the callout in which your code is currently executing.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFRunLoopAddCommonMode` (page 8)

`CFRunLoopCopyAllModes` (page 13)

Declared In
`CFRunLoop.h`

CFRunLoopGetCurrent

Returns the CFRunLoop object for the current thread.

```
CFRunLoopRef CFRunLoopGetCurrent (
    void
);
```

Return Value
Current thread's run loop. Ownership follows the Get Rule.

Discussion
Each thread has exactly one run loop associated with it.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CFRunLoopGetMain` (page 14)

Related Sample Code
CFLocalServer

databurntest

DockBrowser

HID Config Save

ImageClient

Declared In
`CFRunLoop.h`

CFRunLoopGetMain

Returns the main CFRunLoop object.

```
CFRunLoopRef CFRunLoopGetMain (
    void
);
```

Return Value
The main run loop. Ownership follows the Get Rule.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
DispatchFractal

Declared In
`CFRunLoop.h`

CFRunLoopGetNextTimerFireDate

Returns the time at which the next timer will fire.

```
CFAbsoluteTime CFRunLoopGetNextTimerFireDate (
   CFRunLoopRef rl,
   CFStringRef mode
);
```

Parameters

rl

The run loop to examine.

mode

The run loop mode within *rl* to test.

Return Value
The earliest firing time of the run loop timers registered in *mode* for the run loop *rl*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFRunLoop.h`

CFRunLoopGetTypeID

Returns the type identifier for the CFRunLoop opaque type.

```
CFTypeID CFRunLoopGetTypeID (
   void
);
```

Return Value
The type identifier for the CFRunLoop opaque type.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CFLocalServer

Declared In
`CFRunLoop.h`

CFRunLoopIsWaiting

Returns a Boolean value that indicates whether the run loop is waiting for an event.

```
Boolean CFRunLoopIsWaiting (
   CFRunLoopRef rl
);
```

Parameters

rl

The run loop to examine.

Return Value
`true` if *rl* has no events to process and is blocking, waiting for a source or timer to become ready to fire; `false` if *rl* either is not running or is currently processing a source, timer, or observer.

Discussion
This function is useful only to test the state of another thread's run loop. When called with the current thread's run loop, this function always returns `false`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFRunLoop.h`

CFRunLoopPerformBlock

Enqueues a Block on a given runloop to be executed as the runloop cycles in specified modes.

```
void CFRunLoopPerformBlock (
    CFRunLoopRef rl,
    CFTypeRef mode,
    void (^block)(void)
);
```

Parameters

rl

A run loop.

mode

A CFString that identifies a runloop mode, or a CFArray of CFStrings that each identify a runloop mode.

block

The Block to execute.

The Block is copied by the function before the function returns.

Discussion

When the runloop is run in the mode or one of the modes specified by *mode*, the Block is executed. You can use this function as a means to get work "onto" another thread (similar to Cocoa's `performSelector:onThread:withObject:waitUntilDone:` and related methods) as an alternative to mechanisms such as putting a CFRunLoopTimer in the other thread's run loop, or using CFMessagePort to pass information between threads.

Availability

Available in Mac OS X v10.6 and later.

Declared In

`CFRunLoop.h`

CFRunLoopRemoveObserver

Removes a CFRunLoopObserver object from a run loop mode.

```
void CFRunLoopRemoveObserver (
   CFRunLoopRef rl,
   CFRunLoopObserverRef observer,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

observer

The run loop observer to remove.

mode

The run loop mode of *rl* from which to remove *observer*. Use the constant `kCFRunLoopCommonModes` (page 23) to remove *observer* from the set of objects monitored by all the common modes.

Discussion

If *rl* does not contain *observer* in *mode*, this function does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddObserver` (page 8)

`CFRunLoopContainsObserver` (page 11)

Declared In

`CFRunLoop.h`

CFRunLoopRemoveSource

Removes a CFRunLoopSource object from a run loop mode.

```
void CFRunLoopRemoveSource (
   CFRunLoopRef rl,
   CFRunLoopSourceRef source,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

source

The run loop source to remove.

mode

The run loop mode of *rl* from which to remove *source*. Use the constant `kCFRunLoopCommonModes` (page 23) to remove *source* from the set of objects monitored by all the common modes.

Discussion

If *source* is a version 0 source, this function calls the `cancel` callback function specified in the context structure for *source*. See `CFRunLoopSourceContext` and `CFRunLoopSourceContext1`for more details.

If *rl* does not contain *source* in *mode*, this function does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CFRunLoopAddSource` (page 9)

`CFRunLoopContainsSource` (page 11)

Related Sample Code

BackgroundExporter

DNSServiceMetaQuery

HID Utilities Source

iChatStatusFromApplication

SampleUSBMIDIDriver

Declared In

`CFRunLoop.h`

CFRunLoopRemoveTimer

Removes a CFRunLoopTimer object from a run loop mode.

```
void CFRunLoopRemoveTimer (
   CFRunLoopRef rl,
   CFRunLoopTimerRef timer,
   CFStringRef mode
);
```

Parameters

rl

The run loop to modify.

`timer`

The run loop timer to remove.

`mode`

The run loop mode of *`rl`* from which to remove *`timer`*. Use the constant `kCFRunLoopCommonModes` (page 23) to remove *`timer`* from the set of objects monitored by all the common modes.

Discussion
If *`rl`* does not contain *`timer`* in *`mode`*, this function does nothing.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GLUT

SampleUSBMIDIDriver

Declared In
`CFRunLoop.h`

CFRunLoopRun

Runs the current thread's CFRunLoop object in its default mode indefinitely.

```
void CFRunLoopRun (
   void
);
```

Discussion
The current thread's run loop runs in the default mode (see "Default Run Loop Mode" (page 23)) until the run loop is stopped with `CFRunLoopStop` (page 20) or all the sources and timers are removed from the default run loop mode.

Run loops can be run recursively. You can call `CFRunLoopRun` from within any run loop callout and create nested run loop activations on the current thread's call stack.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
audioburntest

bulkerase

CFLocalServer

databurntest

STUCAuthoringDeviceTool

Declared In
`CFRunLoop.h`

CFRunLoopRunInMode

Runs the current thread's CFRunLoop object in a particular mode.

```
SInt32 CFRunLoopRunInMode (
   CFStringRef mode,
   CFTimeInterval seconds,
   Boolean returnAfterSourceHandled
);
```

Parameters

mode

The run loop mode to run. *mode* can be any arbitrary CFString. You do not need to explicitly create a run loop mode, although a run loop mode needs to contain at least one source or timer to run.

seconds

The length of time to run the run loop. If `0`, only one pass is made through the run loop before returning; if multiple sources or timers are ready to fire immediately, only one (possibly two if one is a version 0 source) will be fired, regardless of the value of *returnAfterSourceHandled*.

returnAfterSourceHandled

A flag indicating whether the run loop should exit after processing one source. If `false`, the run loop continues processing events until *seconds* has passed.

Return Value

A value indicating the reason the run loop exited. Possible values are described below.

Discussion

Run loops can be run recursively. You can call `CFRunLoopRunInMode` from within any run loop callout and create nested run loop activations on the current thread's call stack. You are not restricted in which modes you can run from within a callout. You can create another run loop activation running in any available run loop mode, including any modes already running higher in the call stack.

The run loop exits with the following return values under the indicated conditions:

- `kCFRunLoopRunFinished`. The run loop mode *mode* has no sources or timers.
- `kCFRunLoopRunStopped`. The run loop was stopped with `CFRunLoopStop` (page 20).
- `kCFRunLoopRunTimedOut`. The time interval *seconds* passed.
- `kCFRunLoopRunHandledSource`. A source was processed. This exit condition only applies when *returnAfterSourceHandled* is `true`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioQueueTools

DefaultOutputUnit

FSFileOperation

HID Manager Basics

STUCAuthoringDeviceCocoaSample

Declared In

`CFRunLoop.h`

CFRunLoopStop

Forces a CFRunLoop object to stop running.

```
void CFRunLoopStop (
   CFRunLoopRef rl
);
```

Parameters

`rl`

The run loop to stop.

Discussion

This function forces *`rl`* to stop running and return control to the function that called `CFRunLoopRun` (page 19) or `CFRunLoopRunInMode` (page 19) for the current run loop activation. If the run loop is nested with a callout from one activation starting another activation running, only the innermost activation is exited.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

audioburntest

bulkerase

CFFTPSample

CFLocalServer

databurntest

Declared In

`CFRunLoop.h`

CFRunLoopWakeUp

Wakes a waiting CFRunLoop object.

```
void CFRunLoopWakeUp (
   CFRunLoopRef rl
);
```

Parameters

`rl`

The run loop to wake up.

Discussion

A run loop goes to sleep when it is waiting for a source or timer to become ready to fire. If no source or timer fires, the run loop stays there until it times out or is explicitly woken up. If a run loop is modified, such as a new source added, you need to wake up the run loop to allow it to process the change. Version 0 sources use `CFRunLoopWakeUp` to cause the run loop to wake up after setting a source to be signaled, if they want the source handled immediately.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFRunLoop.h`

Data Types

CFRunLoopRef

A reference to a run loop object.

```
typedef struct __CFRunLoop *CFRunLoopRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CFRunLoop.h`

Constants

CFRunLoopRunInMode Exit Codes

Return codes for `CFRunLoopRunInMode`, identifying the reason the run loop exited.

```
enum {
    kCFRunLoopRunFinished = 1,
    kCFRunLoopRunStopped = 2,
    kCFRunLoopRunTimedOut = 3,
    kCFRunLoopRunHandledSource = 4
};
```

Constants

`kCFRunLoopRunFinished`

The running run loop mode has no sources or timers to process.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopRunStopped`

`CFRunLoopStop` (page 20) was called on the run loop.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopRunTimedOut`

The specified time interval for running the run loop has passed.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

`kCFRunLoopRunHandledSource`

A source has been processed. This value is returned only if the run loop was told to run only until a source was processed.

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

Common Mode Flag

A run loop pseudo-mode that manages objects monitored in the "common" modes.

```
const CFStringRef kCFRunLoopCommonModes;
```

Constants

`kCFRunLoopCommonModes`

Objects added to a run loop using this value as the mode are monitored by all run loop modes that have been declared as a member of the set of "common" modes with `CFRunLoopAddCommonMode` (page 8).

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

Discussion

Run loops never run in this mode. This pseudo-mode is used only as a special set of sources, timers, and observers that is shared by other modes. See "Managing Observers" (page 7) for more details.

Default Run Loop Mode

Default run loop mode.

```
const CFStringRef kCFRunLoopDefaultMode;
```

Constants

`kCFRunLoopDefaultMode`

Run loop mode that should be used when a thread is in its default, or idle, state, waiting for an event. This mode is used when the run loop is started with `CFRunLoopRun` (page 19).

Available in Mac OS X v10.0 and later.

Declared in `CFRunLoop.h`.

Document Revision History

This table describes the changes to *CFRunLoop Reference*.

Date	Notes
2009-05-16	Updated for Mac OS X v10.6.
2006-12-08	Updated to include API introduced in Mac OS X v10.5.
2006-02-07	Made formatting changes.
2005-10-04	Corrected duplication of information in the Introduction.
2005-04-29	Moved Introduction to new Introduction page.
2004-04-22	Correction to return values cited in discussion of `CFRunLoopRunInMode`.
2003-01-01	First version of this document.