# CFRunLoop Reference

January 1, 2003

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



Simultaneously published in the United States and Canada.

# Contents

# CFRunLoop Reference

**Derived From:** CFType

**Framework:** CoreFoundation/CoreFoundation.h

**Header:** CFRunLoop.h

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), timers (CFRunLoopTimer), and observers (CFRunLoopObserver). To receive callbacks when these objects need processing, you must first place these objects into a run loop with `CFRunLoopAddSource` (page 12), `CFRunLoopAddTimer` (page 12), or `CFRunLoopAddObserver` (page 11). 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 18), 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 18) 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 10). The default mode, `kCFRunLoopDefaultMode` (page 18), is always a member of the set of common modes. The `kCFRunLoopCommonModes` (page 18) constant is never passed to `CFRunLoopRunInMode` (page 9). 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 `CFRunLoopGetCurrent` (page 7). Call `CFRunLoopRun` (page 8) to run the current thread's run loop in the default mode until the run loop is stopped with `CFRunLoopStop` (page 9). You can also call `CFRunLoopRunInMode` (page 9) 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 8) or `CFRunLoopRunInMode` (page 9) 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.

#### Programming Topics

Run Loops

# Functions

## Using Run Loops

### CFRunLoopCopyAllModes

Returns a list of the defined modes for a CFRunLoop.

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

#### Parameter Descriptions

*rl*

The run loop to use.

*function result* A list of all the run loop modes defined for *rl*. You are responsible for releasing this object.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopCopyCurrentMode

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

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

#### Parameter Descriptions

*rl*

The run loop to use.

*function result* The mode in which *rl* is currently running; NULL if *rl* is not running. You are responsible for releasing this object.

#### 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.

### CFRunLoopGetCurrent

Returns the CFRunLoop object for the current thread.

```
CFRunLoopRef CFRunLoopGetCurrent ();
```

*function result* Current thread's run loop. You are responsible for retaining and releasing this object as needed.

#### Discussion

Each thread has exactly one run loop associated with it.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopGetNextTimerFireDate

Returns the time at which the next timer will fire.

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

#### Parameter Descriptions

*rl*

The run loop to test.

*mode*

The run loop mode within *rl* to test.

*function result* 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.

## CFRunLoopGetTypeID

Returns the type identifier of all CFRunLoop objects.

```
CFTypeID CFRunLoopGetTypeID ();
```

*function result* The type identifier for the CFRunLoop opaque type.

### Availability

Available in Mac OS X v10.0 and later.

## CFRunLoopIsWaiting

Returns whether the run loop is waiting for an event.

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

### Parameter Descriptions

*rl*

The run loop to test.

*function result* `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.

## CFRunLoopRun

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

```
void CFRunLoopRun ();
```

### Discussion

The current thread's run loop runs in the default mode (see "Default Run Loop Mode" (page 18)) until the run loop is stopped with `CFRunLoopStop` (page 9) 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.

## CFRunLoopRunInMode

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

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

### Parameter Descriptions

*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.

*function result* 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:

- kCFRunLoopFinished. The run loop mode *mode* has no sources or timers.
- kCFRunLoopStopped. The run loop was stopped with CFRunLoopStop (page 9).
- kCFRunLoopTimedOut. The time interval *seconds* passed.
- kCFRunLoopHandledSource. A source was processed. This exit condition only applies when *returnAfterSourceHandled* is true.

### Availability

Available in Mac OS X v10.0 and later.

## CFRunLoopStop

Stops a CFRunLoop.

```
void CFRunLoopStop (
```

```
    CFRunLoopRef rl
);
```

#### Parameter Descriptions

*rl*

The run loop to stop.

#### Discussion

This function forces *rl* to stop running and return control to the function that called `CFRunLoopRun` (page 8) or `CFRunLoopRunInMode` (page 9) 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.

### CFRunLoopWakeUp

Wakes up a waiting CFRunLoop.

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

#### Parameter Descriptions

*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.

## Managing Objects in a Run Loop

### CFRunLoopAddCommonMode

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

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

#### Parameter Descriptions

*rl*

The run loop to use. 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 18) for the run loop mode.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopAddObserver

Adds a CFRunLoopObserver to a run loop mode.

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

#### Parameter Descriptions

*rl*

The run loop to use.

*observer*

The run loop observer to add.

*mode*

The run loop mode to which to add *observer*. Use the constant `kCFRunLoopCommonModes` (page 18) 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.

## CFRunLoopAddSource

Adds a CFRunLoopSource to a run loop mode.

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

#### Parameter Descriptions

*rl*

The run loop to use.

*source*

The run loop source to add.

*mode*

The run loop mode to which to add *source*. Use the constant `kCFRunLoopCommonModes` (page 18) 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.

## CFRunLoopAddTimer

Adds a CFRunLoopTimer to a run loop mode.

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

#### Parameter Descriptions

*rl*

The run loop to use.

*timer*

The run loop timer to add.

*mode*

The run loop mode of *rl* to which to add *timer*. Use the constant kCFRunLoopCommonModes (page 18) 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.

## CFRunLoopContainsObserver

Returns whether a run loop mode contains a particular CFRunLoopObserver.

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

#### Parameter Descriptions

*rl*

The run loop to search.

*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 18) to search for *observer* in the set of objects monitored by all the common modes.

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

#### Discussion

If *observer* was added to kCFRunLoopCommonModes (page 18), this function returns `true` if *mode* is either kCFRunLoopCommonModes (page 18) 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.

## CFRunLoopContainsSource

Returns whether a run loop mode contains a particular CFRunLoopSource.

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

#### Parameter Descriptions

*rl*

The run loop to test.

*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 18) to search for *source* in the set of objects monitored by all the common modes.

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

#### Discussion

If *source* was added to `kCFRunLoopCommonModes` (page 18), this function returns `true` if *mode* is either `kCFRunLoopCommonModes` (page 18) 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.

### CFRunLoopContainsTimer

Returns whether a run loop mode contains a particular CFRunLoopTimer.

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

#### Parameter Descriptions

*rl*

The run loop to test.

*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 18) to search for *timer* in the set of objects monitored by all the common modes.

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

### Discussion

If *timer* was added to kCFRunLoopCommonModes (page 18), this function returns true if *mode* is either kCFRunLoopCommonModes (page 18) 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.

## CFRunLoopRemoveObserver

Removes a CFRunLoopObserver from a run loop mode.

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

### Parameter Descriptions

*rl*

The run loop to use.

*observer*

The run loop observer to remove.

*mode*

The run loop mode of *rl* from which to remove *observer*. Use the constant kCFRunLoopCommonModes (page 18) 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.

## CFRunLoopRemoveSource

Removes a CFRunLoopSource from a run loop mode.

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

### Parameter Descriptions

*rl*

The run loop to use.

*source*

The run loop source to remove.

*mode*

The run loop mode of *rl* from which to remove *source*. Use the constant kCFRunLoopCommonModes (page 18) 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.

## CFRunLoopRemoveTimer

Removes a CFRunLoopTimer from a run loop mode.

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

#### Parameter Descriptions

*rl*

The run loop to use.

*timer*

The run loop timer to remove.

*mode*

The run loop mode of *rl* from which to remove *timer*. Use the constant kCFRunLoopCommonModes (page 18) 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.

# Data Types

## Miscellaneous

### CFRunLoopRef

A reference to a run loop object.

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

# Constants

## Miscellaneous

### CFRunLoopRunInMode Exit Codes

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

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

#### Constant Descriptions

`kCFRunLoopRunFinished`

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

`kCFRunLoopRunStopped`

`CFRunLoopStop` (page 9) was called on the run loop.

`kCFRunLoopRunTimedOut`

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

`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.

### Common Mode Flag

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

```
const CFStringRef kCFRunLoopCommonModes;
```

#### Descriptions

`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 10).

#### 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 Objects in a Run Loop" (page 10) for more details.

### Default Run Loop Mode

Default run loop mode.

```
const CFStringRef kCFRunLoopDefaultMode;
```

#### Descriptions

`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 8).

# Document Revision History

Table RH-1 describes the revisions to this document.

**Table RH-1**

| Date | Notes |
|---|---|
| January 2003 | First version of this document. |