# CFRunLoopSource 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

# CFRunLoopSource Reference

**Derived From:** CFType

**Framework:** CoreFoundation/CoreFoundation.h

**Header:** CFRunLoop.h

A CFRunLoopSource is an abstraction of an input source that can be put into a run loop. Input sources typically generate asynchronous events, such as messages arriving on a network port or actions performed by the user.

An input source type normally defines an API for creating and operating on objects of the type, as if it were a separate entity from the run loop, then provides a function to create a CFRunLoopSource for an object. The run loop source can then be registered with the run loop and act as an intermediary between the run loop and the actual input source type object. Examples of input sources include CFMachPort, CFMessagePort, and CFSocket.

There are two categories of sources. Version 0 sources, so named because the `version` field of their context structure is 0, are managed manually by the application. When a source is ready to fire, some part of the application, perhaps code on a separate thread waiting for an event, must call `CFRunLoopSourceSignal` (page 8) to tell the run loop that the source is ready to fire. The run loop source for CFSocket is currently implemented as a version 0 source.

Version 1 sources are managed by the run loop and kernel. These sources use Mach ports to signal when the sources are ready to fire. A source is automatically signaled by the kernel when a message arrives on the source's Mach port. The contents of the message are given to the source to process when the source is fired. The run loop sources for CFMachPort and CFMessagePort are currently implemented as version 1 sources.

When creating your own custom run loop source, you can choose which version works best for you.

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. Adding a source to multiple threads' run loops can be used to manage a pool of "worker" threads that is processing discrete sets of data, such as client-server messages over a network or entries in a job queue filled by a "manager" thread. As messages arrive or jobs get added to the queue, the source gets signaled and a random thread receives and processes the request.

**Programming Topics**

Run Loops

# Functions

## CFRunLoopSourceCreate

Creates a CFRunLoopSource object.

```
CFRunLoopSourceRef CFRunLoopSourceCreate (
    CFAllocatorRef allocator,
    CFIndex order,
    CFRunLoopSourceContext *context
);
```

### Parameter Descriptions

*allocator*

The CFAllocator object to be used to allocate memory for the CFRunLoopSource object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

*order*

A priority index indicating the order in which run loop sources are processed. When multiple run loop sources are firing in a single pass through the run loop, the sources are processed in increasing order of this parameter. If the run loop is set to process only one source per loop, only the highest priority source, the one with the lowest *order* value, is processed. This value is ignored for version 1 sources. Pass 0 unless there is a reason to do otherwise.

*context*

A structure holding contextual information for the run loop source. The function copies the information out of the structure, so the memory pointed to by *context* does not need to persist beyond the function call.

*function result* The new CFRunLoopSource object. You are responsible for releasing this object.

### Discussion

The run loop source is not automatically added to a run loop. To add the CFRunLoopSource object to a run loop, use `CFRunLoopAddSource`.

### Availability

Available in Mac OS X v10.0 and later.

## CFRunLoopSourceGetContext

Returns the context information for a CFRunLoopSource.

```
void CFRunLoopSourceGetContext (
    CFRunLoopSourceRef source,
    CFRunLoopSourceContext *context
);
```

#### Parameter Descriptions

*source*

The run loop source to use.

*context*

A pointer to the structure into which the context information for *source* is to be copied. The information being returned is the same information passed to CFRunLoopSourceCreate (page 6) when creating *source*.

#### Discussion

Run loop sources come in two versions with different-sized context structures. *context* must point to the correct version of the structure for *source*. Before calling this function, you need to initialize the version member of *context* with the version number (either 0 or 1) of *source*.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopSourceGetOrder

Returns the ordering parameter for a CFRunLoopSource.

```
CFIndex CFRunLoopSourceGetOrder (
    CFRunLoopSourceRef source
);
```

#### Parameter Descriptions

*source*

The run loop source to use.

*function result* The ordering parameter for *source*, which the run loop uses (for version 0 sources only) to determine the order in which sources are processed when multiple sources are firing.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopSourceGetTypeID

Returns the type identifier of all CFRunLoopSource objects.

```
CFTypeID CFRunLoopSourceGetTypeID ();
```

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

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopSourceInvalidate

Invalidates a CFRunLoopSource, stopping it from ever firing again.

```
void CFRunLoopSourceInvalidate (
```

```
    CFRunLoopSourceRef source
);
```

#### Parameter Descriptions

*source*

The run loop source to invalidate.

#### Discussion

Once invalidated, *source* will never fire and call its perform callback function again. This function automatically removes *source* from all the run loop modes in which it was registered. If *source* is a version 0 source, this function calls its `cancel` callback function as it is removed from each run loop mode. The memory for *source* is not deallocated unless the run loop held the only reference to *source*.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopSourceIsValid

Returns whether a CFRunLoopSource is valid and able to fire.

```
Boolean CFRunLoopSourceIsValid (
    CFRunLoopSourceRef source
);
```

#### Parameter Descriptions

*source*

The run loop source to test.

*function result* `true` if *source* is valid; `false` if *source* has been invalidated.

#### Availability

Available in Mac OS X v10.0 and later.

### CFRunLoopSourceSignal

Signals a CFRunLoopSource, marking it as ready to fire.

```
void CFRunLoopSourceSignal (
    CFRunLoopSourceRef source
);
```

#### Parameter Descriptions

*source*

The run loop source to signal.

#### Discussion

This function has no effect on version 1 sources, which are automatically handled when Mach messages arrive for them. After signaling a version 0 source, you need to call `CFRunLoopWakeUp` on one of the run loops in which the source is registered to get the source handled immediately.

#### Availability

Available in Mac OS X v10.0 and later.

# Callbacks

### CFRunLoopCancelCallBack

Callback invoked when a version 0 CFRunLoopSource is removed from a run loop mode.

```
typedef void (*CFRunLoopCancelCallBack) (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

#### Parameter Descriptions

*info*

The `info` member of the `CFRunLoopSourceContext` (page 13) structure that was used when creating the run loop source.

*rl*

The run loop from which the run loop source is being removed.

*mode*

The run loop mode from which the run loop source is being removed.

#### Discussion

You specify this callback in the `CFRunLoopSourceContext` (page 13) structure when creating the run loop source.

### CFRunLoopEqualCallBack

Callback invoked to test two CFRunLoopSources for equality.

```
typedef Boolean (*CFRunLoopEqualCallBack) (
    const void *info1,
    const void *info2
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
Boolean MyCallBack (
    const void *info1,
    const void *info2
```

```
);
```

#### Parameter Descriptions

*info1*

The `info` member of the `CFRunLoopSourceContext` (page 13) or `CFRunLoopSourceContext1` (page 14) structure that was used when creating the first run loop source to test.

*info2*

The `info` member of the `CFRunLoopSourceContext` (page 13) or `CFRunLoopSourceContext1` (page 14) structure that was used when creating the second run loop source to test.

*function result* `true` if *info1* and *info2* should be considered equal; `false` otherwise.

#### Discussion

You specify this callback in the `CFRunLoopSourceContext` (page 13) or `CFRunLoopSourceContext1` (page 14) structure when creating the run loop source.

### CFRunLoopGetPortCallBack

Callback invoked to obtain the native Mach port represented by a version 1 CFRunLoopSource.

```
typedef mach_port_t (*CFRunLoopGetPortCallBack) (
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
mach_port_t MyCallBack (
    void *info
);
```

#### Parameter Descriptions

*info*

The `info` member of the `CFRunLoopSourceContext1` (page 14) structure that was used when creating the run loop source.

*function result* The native Mach port for the run loop source.

#### Discussion

This callback is called whenever the run loop needs a source's Mach port, which can happen in each iteration of the run loop's loop. Because of the frequency with which the run loop may call this callback, make the function as efficient as possible.

A version 1 run loop source must have a one-to-one relationship between itself and its Mach port. Each source must have only one Mach port associated with it and each Mach port must represent only one source.

You specify this callback in the `CFRunLoopSourceContext1` (page 14) structure when creating the run loop source.

### CFRunLoopHashCallBack

Callback invoked to compute a hash code for the `info` pointer of a CFRunLoopSource.

```
typedef CFHashCode (*CFRunLoopHashCallBack) (
    const void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
CFHashCode MyCallBack (
    const void *info
);
```

#### Parameter Descriptions

*info*

The `info` member of the `CFRunLoopSourceContext` (page 13) or `CFRunLoopSourceContext1` (page 14) structure that was used when creating the run loop source.

*function result* A hash code value for *info*.

#### Discussion

If a hash callback is not provided for a source, the `info` pointer is used.

You specify this callback in the `CFRunLoopSourceContext` (page 13) or `CFRunLoopSourceContext1` (page 14) structure when creating the run loop source.

### CFRunLoopMachPerformCallBack

Callback invoked to process and optionally reply to a message received on a version 1 CFRunLoopSource (Mach port-based sources).

```
typedef void *(*CFRunLoopMachPerformCallBack) (
    void *msg,
    CFIndex size,
    CFAllocatorRef allocator,
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void *MyCallBack (
    void *msg,
    CFIndex size,
    CFAllocatorRef allocator,
    void *info
);
```

#### Parameter Descriptions

*msg*

The Mach message received on the Mach port. The pointer is to a `mach_msg_header_t` structure. A version 0 format trailer (`mach_msg_format_0_trailer_t`) is at the end of the Mach message.

*size*

Size of the Mach message in *msg*, excluding the message trailer.

*allocator*

The allocator object that should be used to allocate a reply message.

*info*

The `info` member of the `CFRunLoopSourceContext1` (page 14) structure that was used when creating the run loop source.

*function result* An optional Mach message to be sent in response to the received message. The message must be allocated using *allocator*. Return `NULL` if you want an empty reply returned to the sender.

#### Discussion

You only need to provide this callback if you create your own version 1 run loop source. CFMachPort and CFMessagePort run loop sources already implement this callback to forward the received message to the CFMachPort's or CFMessagePort's own callback function, which you do need to implement.

You specify this callback in the `CFRunLoopSourceContext1` (page 14) structure when creating the run loop source.

### CFRunLoopPerformCallBack

Callback invoked when a message is received on a version 0 CFRunLoopSource.

```
typedef void (*CFRunLoopPerformCallBack) (
    void *info
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info
);
```

#### Parameter Descriptions

*info*

The `info` member of the `CFRunLoopSourceContext` (page 13) structure that was used when creating the run loop source.

#### Discussion

You only need to provide this callback if you create your own version 0 run loop source. CFSocket run loop sources already implement this callback to forward the received message to the CFSocket's own callback function, which you do need to implement.

You specify this callback in the `CFRunLoopSourceContext` (page 13) structure when creating the run loop source.

### CFRunLoopScheduleCallBack

Callback invoked when a version 0 CFRunLoopSource is added to a run loop mode.

```
typedef void (*CFRunLoopScheduleCallBack) (
    void *info,
```

```
    CFRunLoopRef rl,
    CFStringRef mode
);
```

If you name your function `MyCallBack`, you would declare it like this:

```
void MyCallBack (
    void *info,
    CFRunLoopRef rl,
    CFStringRef mode
);
```

#### Parameter Descriptions

*info*

The `info` member of the `CFRunLoopSourceContext` (page 13) structure that was used when creating the run loop source.

*rl*

The run loop in which the source is being scheduled.

*mode*

The run loop mode in which the source is being scheduled.

#### Discussion

You specify this callback in the `CFRunLoopSourceContext` (page 13) structure when creating the run loop source.

# Data Types

## Miscellaneous

### CFRunLoopSourceContext

A structure that contains program-defined data and callbacks with which you can configure a version 0 CFRunLoopSource's behavior.

```
struct CFRunLoopSourceContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFRunLoopEqualCallBack equal;
    CFRunLoopHashCallBack hash;
    CFRunLoopScheduleCallBack schedule;
    CFRunLoopCancelCallBack cancel;
    CFRunLoopPerformCallBack perform;
};
typedef struct CFRunLoopSourceContext CFRunLoopSourceContext;
```

#### Field Descriptions

`version`

Version number of the structure. Must be 0.

`info`

An arbitrary pointer to program-defined data, which can be associated with the CFRunLoopSource at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`

A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`

A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`

A copy description callback for your program-defined `info` pointer. Can be `NULL`.

`equal`

An equality test callback for your program-defined `info` pointer. Can be `NULL`.

`hash`

A hash calculation callback for your program-defined `info` pointer. Can be `NULL`.

`schedule`

A scheduling callback for the run loop source. This callback is called when the source is added to a run loop mode. Can be `NULL`.

`cancel`

A cancel callback for the run loop source. This callback is called when the source is removed from a run loop mode. Can be `NULL`.

`perform`

A perform callback for the run loop source. This callback is called when the source has fired.

### CFRunLoopSourceContext1

A structure that contains program-defined data and callbacks with which you can configure a version 1 CFRunLoopSource's behavior.

```
struct CFRunLoopSourceContext1 {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFRunLoopEqualCallBack equal;
    CFRunLoopHashCallBack hash;
    CFRunLoopGetPortCallBack getPort;
    CFRunLoopMachPerformCallBack perform;
};
typedef struct CFRunLoopSourceContext1 CFRunLoopSourceContext1;
```

#### Field Descriptions

`version`

Version number of the structure. Must be 1.

`info`

An arbitrary pointer to program-defined data, which can be associated with the run loop source at creation time. This pointer is passed to all the callbacks defined in the context.

`retain`

A retain callback for your program-defined `info` pointer. Can be `NULL`.

`release`

A release callback for your program-defined `info` pointer. Can be `NULL`.

`copyDescription`

A copy description callback for your program-defined `info` pointer. Can be `NULL`.

`equal`

An equality test callback for your program-defined `info` pointer. Can be `NULL`.

`hash`

A hash calculation callback for your program-defined `info` pointer. Can be `NULL`.

`getPort`

A callback to retrieve the native Mach port represented by the source. This callback is called when the source is either added to or removed from a run loop mode.

`perform`

A perform callback for the run loop source. This callback is called when the source has fired.

### `CFRunLoopSourceRef`

A reference to a run loop source object.

```
typedef struct __CFRunLoopSource *CFRunLoopSourceRef;
```

# Document Revision History

Table RH-1 describes the revisions to this document.

**Table RH-1**

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

Document Revision History