CFSocket Reference

Core Foundation

2008-10-15

Contents

CFSocket Reference

Derived From:	CFType
Framework:	CoreFoundation/CoreFoundation.h
Declared in	CFSocket.h
Companion guides	CFNetwork Programming Guide Threading Programming Guide

Overview

A CFSocket is a communications channel implemented with a BSD socket.

CFSockets can be created from scratch (`CFSocketCreate` (page 8) and `CFSocketCreateWithSocketSignature` (page 11)), from a pre-existing BSD socket (`CFSocketCreateWithNative` (page 11)), or already connected to a remote socket (`CFSocketCreateConnectedToSocketSignature` (page 9)).

To listen for messages, you need to create a run loop source with `CFSocketCreateRunLoopSource` (page 10) and add it to a run loop with `CFRunLoopAddSource`. You can select the types of socket activities, such as connection attempts or data arrivals, that cause the source to fire and invoke your CFSocket's callback function. To send data, you store the data in a CFData and call `CFSocketSendData` (page 16).

Unlike Mach and message ports, sockets support communication over a network.

Functions by Task

Creating Sockets

`CFSocketCreate` (page 8)
: Creates a CFSocket object of a specified protocol and type.

`CFSocketCreateConnectedToSocketSignature` (page 9)
: Creates a CFSocket object and opens a connection to a remote socket.

`CFSocketCreateWithNative` (page 11)
: Creates a CFSocket object for a pre-existing native socket.

`CFSocketCreateWithSocketSignature` (page 11)
: Creates a CFSocket object using information from a CFSocketSignature structure.

Configuring Sockets

`CFSocketCopyAddress` (page 7)
: Returns the local address of a CFSocket object.

`CFSocketCopyPeerAddress` (page 7)
: Returns the remote address to which a CFSocket object is connected.

`CFSocketDisableCallBacks` (page 12)
: Disables the callback function of a CFSocket object for certain types of socket activity.

`CFSocketEnableCallBacks` (page 13)
: Enables the callback function of a CFSocket object for certain types of socket activity.

`CFSocketGetContext` (page 13)
: Returns the context information for a CFSocket object.

`CFSocketGetNative` (page 14)
: Returns the native socket associated with a CFSocket object.

`CFSocketGetSocketFlags` (page 14)
: Returns flags that control certain behaviors of a CFSocket object.

`CFSocketSetAddress` (page 17)
: Binds a local address to a CFSocket object.

`CFSocketSetSocketFlags` (page 18)
: Sets flags that control certain behaviors of a CFSocket object.

Using Sockets

`CFSocketConnectToAddress` (page 6)
: Opens a connection to a remote socket.

`CFSocketCreateRunLoopSource` (page 10)
: Creates a CFRunLoopSource object for a CFSocket object.

`CFSocketGetTypeID` (page 15)
: Returns the type identifier for the CFSocket opaque type.

`CFSocketInvalidate` (page 15)
: Invalidates a CFSocket object, stopping it from sending or receiving any more messages.

`CFSocketIsValid` (page 16)
: Returns a Boolean value that indicates whether a CFSocket object is valid and able to send or receive messages.

`CFSocketSendData` (page 16)
: Sends data over a CFSocket object.

Functions

CFSocketConnectToAddress

Opens a connection to a remote socket.

```
CFSocketError CFSocketConnectToAddress (
   CFSocketRef s,
   CFDataRef address,
   CFTimeInterval timeout
);
```

Parameters

s

The CFSocket object with which to connect to *address*.

address

A CFData object containing a `struct sockaddr` appropriate for the protocol family of *s*, indicating the remote address to which to connect.

timeout

The time to wait for a connection to succeed. If a negative value is used, this function does not wait for the connection and instead lets the connection attempt happen in the background. If *s* requested a `kCFSocketConnectCallBack`, you will receive a callback when the background connection succeeds or fails.

Return Value

An error code indicating success or failure of the connection attempt.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketCopyAddress

Returns the local address of a CFSocket object.

```
CFDataRef CFSocketCopyAddress (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value

The local address, stored as a `struct sockaddr` in a CFData object, of *s*. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketCopyPeerAddress

Returns the remote address to which a CFSocket object is connected.

```
CFDataRef CFSocketCopyPeerAddress (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value

The remote address, stored as a `struct sockaddr` in a CFData object, to which *s* is connected. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketCreate

Creates a CFSocket object of a specified protocol and type.

```
CFSocketRef CFSocketCreate (
   CFAllocatorRef allocator,
   SInt32 protocolFamily,
   SInt32 socketType,
   SInt32 protocol,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

protocolFamily

The protocol family for the socket. If negative or 0 is passed, the socket defaults to `PF_INET`.

socketType

The socket type to create. If *protocolFamily* is `PF_INET` and *socketType* is negative or 0, the socket type defaults to `SOCK_STREAM`.

protocol

The protocol for the socket. If *protocolFamily* is `PF_INET` and *protocol* is negative or 0, the socket protocol defaults to `IPPROTO_TCP` if *socketType* is `SOCK_STREAM` or `IPPROTO_UDP` if *socketType* is `SOCK_DGRAM`.

callBackTypes

A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 21) for the possible activity values.

callout

The function to call when one of the activities indicated by *callBackTypes* occurs.

context

A structure holding contextual information for the CFSocket object. 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. Can be `NULL`.

Return Value

The new CFSocket object, or `NULL` if an error occurred. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CocoaEcho

CocoaHTTPServer

CocoaSOAP

Declared In

`CFSocket.h`

CFSocketCreateConnectedToSocketSignature

Creates a CFSocket object and opens a connection to a remote socket.

```
CFSocketRef CFSocketCreateConnectedToSocketSignature (
   CFAllocatorRef allocator,
   const CFSocketSignature *signature,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context,
   CFTimeInterval timeout
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

signature

A `CFSocketSignature` (page 21) identifying the communication protocol and address to which the CFSocket object should connect.

callBackTypes

A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 21) for the possible activity values.

callout

The function to call when one of the activities indicated by *callBackTypes* occurs.

context

A structure holding contextual information for the CFSocket object. 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. Can be `NULL`.

timeout

The time to wait for a connection to succeed. If a negative value is used, this function does not wait for the connection and instead lets the connection attempt happen in the background. If *callBackTypes* includes `kCFSocketConnectCallBack`, you will receive a callback when the background connection succeeds or fails.

Return Value

The new CFSocket object, or `NULL` if an error occurred. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketCreateRunLoopSource

Creates a CFRunLoopSource object for a CFSocket object.

```
CFRunLoopSourceRef CFSocketCreateRunLoopSource (
   CFAllocatorRef allocator,
   CFSocketRef s,
   CFIndex order
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

s

The CFSocket object for which to create a run loop source.

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.

Return Value

The new CFRunLoopSource object for *s*. Ownership follows the Create Rule.

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFLocalServer

CocoaEcho

CocoaHTTPServer

CocoaSOAP

DNSServiceMetaQuery

Declared In
`CFSocket.h`

CFSocketCreateWithNative

Creates a CFSocket object for a pre-existing native socket.

```
CFSocketRef CFSocketCreateWithNative (
   CFAllocatorRef allocator,
   CFSocketNativeHandle sock,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

`allocator`
: The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

`sock`
: The native socket for which to create a CFSocket object.

`callBackTypes`
: A bitwise-OR combination of the types of socket activity that should cause *`callout`* to be called. See Callback Types (page 21) for the possible activity values.

`callout`
: The function to call when one of the activities indicated by *`callBackTypes`* occurs.

`context`
: A structure holding contextual information for the CFSocket object. 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. Can be `NULL`.

Return Value
The new CFSocket object, or `NULL` if an error occurred. If a CFSocket object already exists for *`sock`*, the function returns the pre-existing object instead of creating a new object; the *`context`*, *`callout`*, and *`callBackTypes`* parameters are ignored in this case. Ownership follows the Create Rule.

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

Related Sample Code
CFLocalServer

DNSServiceMetaQuery

Declared In
`CFSocket.h`

CFSocketCreateWithSocketSignature

Creates a CFSocket object using information from a CFSocketSignature structure.

```
CFSocketRef CFSocketCreateWithSocketSignature (
   CFAllocatorRef allocator,
   const CFSocketSignature *signature,
   CFOptionFlags callBackTypes,
   CFSocketCallBack callout,
   const CFSocketContext *context
);
```

Parameters

allocator
: The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

signature
: A `CFSocketSignature` (page 21) identifying the communication protocol and address with which to create the CFSocket object.

callBackTypes
: A bitwise-OR combination of the types of socket activity that should cause *callout* to be called. See Callback Types (page 21) for the possible activity values.

callout
: The function to call when one of the activities indicated by *callBackTypes* occurs.

context
: A structure holding contextual information for the CFSocket object. 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. Can be `NULL`.

Return Value

The new CFSocket object, or `NULL` if an error occurred. Ownership follows the Create Rule.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketDisableCallBacks

Disables the callback function of a CFSocket object for certain types of socket activity.

```
void CFSocketDisableCallBacks (
   CFSocketRef s,
   CFOptionFlags callBackTypes
);
```

Parameters

s
: The CFSocket object to modify.

callBackTypes
: A bitwise-OR combination of CFSocket activity types that should not cause the callback function of *s* to be called. See Callback Types (page 21) for a list of callback types.

Discussion

If you no longer want certain types of callbacks that you requested when creating *s*, you can use this function to temporarily disable the callback. Use `CFSocketEnableCallBacks` (page 13) to reenable a callback type.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CFSocket.h`

CFSocketEnableCallBacks

Enables the callback function of a CFSocket object for certain types of socket activity.

```
void CFSocketEnableCallBacks (
   CFSocketRef s,
   CFOptionFlags callBackTypes
);
```

Parameters

s

The CFSocket object to modify.

callBackTypes

A bitwise-OR combination of CFSocket activity types that should cause the callback function of *s* to be called. See Callback Types (page 21) for a list of callback types.

Discussion
If a callback type is not automatically reenabled, you can use this function to enable the callback. A callback type that is not automatically reenabled still does not get reenabled after enabling it with this function; use `CFSocketSetSocketFlags` (page 18) to have the callback type reenabled automatically.

Be sure to enable only callback types that your CFSocket object actually possesses and has requested when creating the CFSocket object; the result of enabling other callback types is undefined.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
CFLocalServer

Declared In
`CFSocket.h`

CFSocketGetContext

Returns the context information for a CFSocket object.

```
void CFSocketGetContext (
   CFSocketRef s,
   CFSocketContext *context
);
```

Parameters

s

The CFSocket object to examine.

`context`

A pointer to the structure into which the context information for *`s`* is to be copied. The information being returned is usually the same information you passed to `CFSocketCreate` (page 8), `CFSocketCreateConnectedToSocketSignature` (page 9), `CFSocketCreateWithNative` (page 11), or `CFSocketCreateWithSocketSignature` (page 11) when creating the CFSocket object. However, if `CFSocketCreateWithNative` (page 11) returned a cached CFSocket object instead of creating a new object, *`context`* is filled with information from the original CFSocket object instead of the information you passed to the function.

Discussion

The context version number for CFSocket is currently 0. Before calling this function, you need to initialize the `version` member of *`context`* to 0.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketGetNative

Returns the native socket associated with a CFSocket object.

```
CFSocketNativeHandle CFSocketGetNative (
   CFSocketRef s
);
```

Parameters

`s`

The CFSocket object to examine.

Return Value

The native socket associated with *`s`*. If *`s`* has been invalidated, returns `-1`, `INVALID_SOCKET`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CFLocalServer

CocoaEcho

CocoaHTTPServer

CocoaSOAP

Declared In

`CFSocket.h`

CFSocketGetSocketFlags

Returns flags that control certain behaviors of a CFSocket object.

```
CFOptionFlags CFSocketGetSocketFlags (
   CFSocketRef s
);
```

Parameters

s

The CFSocket to examine.

Return Value

A bitwise-OR combination of flags controlling the behavior of *s*. See CFSocket Flags (page 23) for the list of available flags.

Discussion

See `CFSocketSetSocketFlags` (page 18) for details on what the flags of a CFSocket mean.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

DNSServiceMetaQuery

Declared In

`CFSocket.h`

CFSocketGetTypeID

Returns the type identifier for the CFSocket opaque type.

```
CFTypeID CFSocketGetTypeID ();
```

Return Value

The type identifier for the CFSocket opaque type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CFSocket.h`

CFSocketInvalidate

Invalidates a CFSocket object, stopping it from sending or receiving any more messages.

```
void CFSocketInvalidate (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to invalidate.

Discussion
Invalidating a CFSocket object prevents the port from ever sending or receiving any more messages. The CFSocket object is not deallocated, though. The `CFSocketContext` (page 20) `info` information, which was provided when *s* was created, is released, if a release callback was specified in its context structure. Also, if a run loop source was created for *s*, the run loop source is invalidated, as well.

You should always invalidate a socket when you are done using it. If you have requested, using `CFSocketSetSocketFlags` (page 18), that the underlying socket not automatically close when invalidating the wrapping CFSocket object, you must invalidate the CFSocket object before closing the socket yourself.

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

Related Sample Code
CFLocalServer

CocoaEcho

CocoaHTTPServer

CocoaSOAP

DNSServiceMetaQuery

Declared In
`CFSocket.h`

CFSocketIsValid

Returns a Boolean value that indicates whether a CFSocket object is valid and able to send or receive messages.

```
Boolean CFSocketIsValid (
   CFSocketRef s
);
```

Parameters

s

The CFSocket object to examine.

Return Value
`true` if *s* can be used for communication, otherwise `false`.

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

Declared In
`CFSocket.h`

CFSocketSendData

Sends data over a CFSocket object.

```
CFSocketError CFSocketSendData (
   CFSocketRef s,
   CFDataRef address,
   CFDataRef data,
   CFTimeInterval timeout
);
```

Parameters

s

The CFSocket object to use.

address

The address, stored as a `struct sockaddr` in a CFData object, to which to send the contents of *data*. If `NULL`, the data are sent to the address to which *s* is already connected.

data

The data to send.

timeout

The time to wait for the data to be sent.

Return Value
An error code indicating success or failure.

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

Declared In
`CFSocket.h`

CFSocketSetAddress

Binds a local address to a CFSocket object.

```
CFSocketError CFSocketSetAddress (
   CFSocketRef s,
   CFDataRef address
);
```

Parameters

s

The CFSocket object to modify.

address

A CFData object containing a `struct sockaddr` appropriate for the protocol family of *s*.

Return Value
An error code indicating success or failure.

Discussion
Once *s* is bound to *address*, depending on the socket's protocol, other processes and computers can connect to *s*.

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

Related Sample Code
CocoaEcho

CocoaHTTPServer
CocoaSOAP

Declared In
`CFSocket.h`

CFSocketSetSocketFlags

Sets flags that control certain behaviors of a CFSocket object.

```
void CFSocketSetSocketFlags (
   CFSocketRef s,
   CFOptionFlags flags
);
```

Parameters

`s`
The CFSocket object to modify.

`flags`
A bitwise-OR combination of flags controlling the behavior of *`s`*. See CFSocket Flags (page 23) for the list of available flags.

Discussion
The *`flags`* argument controls whether callbacks of a given type are automatically reenabled after they are triggered, and whether the underlying native socket is closed when *`s`* is invalidated.

By default `kCFSocketReadCallBack`, `kCFSocketAcceptCallBack`, and `kCFSocketDataCallBack` callbacks are automatically reenabled, whereas `kCFSocketWriteCallBack` callbacks are not; `kCFSocketConnectCallBack` callbacks can only occur once, so they cannot be reenabled. Be careful about automatically re-enabling read and write callbacks, because this implies that the callbacks will be sent repeatedly if the socket remains readable or writable respectively. Be sure to set these flags only for callback types that your CFSocket object actually possesses; the result of setting them for other callback types is undefined.

By default the underlying native socket will be closed when *`s`* is invalidated, but it will not be if the `kCFSocketCloseOnInvalidate` flag is turned off. This can be useful when you want to destroy a CFSocket object but continue to use the underlying native socket. The CFSocket object must still be invalidated when it will no longer be used. Do not in either case close the underlying native socket without invalidating the CFSocket object.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
DNSServiceMetaQuery

Declared In
`CFSocket.h`

Callbacks

CFSocketCallBack

Callback invoked when certain types of activity takes place on a CFSocket object.

```
typedef void (*CFSocketCallBack) (
    CFSocketRef s,
    CFSocketCallBackType callbackType,
    CFDataRef address,
    const void *data,
    void *info
);
```

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

```
void MyCallBack (
    CFSocketRef s,
    CFSocketCallBackType callbackType,
    CFDataRef address,
    const void *data,
    void *info
);
```

Parameters

`s`

The CFSocket object that experienced some activity.

`callbackType`

The type of activity detected.

`address`

A CFData object holding the contents of a `struct sockaddr` appropriate for the protocol family of *`s`*, identifying the remote address to which *`s`* is connected. This value is `NULL` except for `kCFSocketAcceptCallBack` and `kCFSocketDataCallBack` callbacks.

`data`

Data appropriate for the callback type. For a `kCFSocketConnectCallBack` that failed in the background, it is a pointer to an `SInt32` error code; for a `kCFSocketAcceptCallback`, it is a pointer to a `CFSocketNativeHandle` (page 20); or for a `kCFSocketDataCallBack`, it is a CFData object containing the incoming data. In all other cases, it is `NULL`.

`info`

The `info` member of the `CFSocketContext` (page 20) structure that was used when creating the CFSocket object.

Discussion

You specify this callback when you create the CFSocket object with `CFSocketCreate` (page 8), `CFSocketCreateConnectedToSocketSignature` (page 9), `CFSocketCreateWithNative` (page 11), or `CFSocketCreateWithSocketSignature` (page 11).

Availability

Available in Mac OS X v10.0 and later.

Declared In
`CFSocket.h`

Data Types

CFSocketContext

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

```
struct CFSocketContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFSocketContext CFSocketContext;
```

Fields

`version`
: Version number of the structure. Must be `0`.

`info`
: An arbitrary pointer to program-defined data, which can be associated with the CFSocket object 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`.

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

Declared In
`CFSocket.h`

CFSocketNativeHandle

Type for the platform-specific native socket handle.

```
typedef int CFSocketNativeHandle;
```

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

Declared In
`CFSocket.h`

CFSocketRef

A reference to a CFSocket object.

```
typedef struct __CFSocket *CFSocketRef;
```

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

Declared In
`CFSocket.h`

CFSocketSignature

A structure that fully specifies the communication protocol and connection address of a CFSocket object.

```
struct CFSocketSignature {
    SInt32 protocolFamily;
    SInt32 socketType;
    SInt32 protocol;
    CFDataRef address;
};
typedef struct CFSocketSignature CFSocketSignature;
```

Fields

`protocolFamily`
    The protocol family of the socket.

`socketType`
    The socket type of the socket.

`protocol`
    The protocol type of the socket.

`address`
    A CFData object holding the contents of a `struct sockaddr` appropriate for the given protocol family, identifying the address of the socket.

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

Declared In
`CFSocket.h`

Constants

Callback Types

Types of socket activity that can cause the callback function of a CFSocket object to be called.

```
enum CFSocketCallBackType {
    kCFSocketNoCallBack = 0,
    kCFSocketReadCallBack = 1,
    kCFSocketAcceptCallBack = 2,
    kCFSocketDataCallBack = 3,
    kCFSocketConnectCallBack = 4,
    kCFSocketWriteCallBack = 8
};
typedef enum CFSocketCallBackType CFSocketCallBackType;
```

Constants

`kCFSocketNoCallBack`

No callback should be made for any activity.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketReadCallBack`

The callback is called when data is available to be read or a new connection is waiting to be accepted. The data is not automatically read; the callback must read the data itself.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketAcceptCallBack`

New connections will be automatically accepted and the callback is called with the data argument being a pointer to a `CFSocketNativeHandle` (page 20) of the child socket. This callback is usable only with listening sockets.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketDataCallBack`

Incoming data will be read in chunks in the background and the callback is called with the data argument being a CFData object containing the read data.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketConnectCallBack`

If a connection attempt is made in the background by calling `CFSocketConnectToAddress` (page 6) or `CFSocketCreateConnectedToSocketSignature` (page 9) with a negative timeout value, this callback type is made when the connect finishes. In this case the data argument is either `NULL` or a pointer to an `SInt32` error code, if the connect failed. This callback will never be sent more than once for a given socket.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketWriteCallBack`

The callback is called when the socket is writable. This callback type may be useful when large amounts of data are being sent rapidly over the socket and you want a notification when there is space in the kernel buffers for more data.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

Discussion

The callback types for which a callback is made is determined when the CFSocket object is created, such as with `CFSocketCreate` (page 8), or later with `CFSocketEnableCallBacks` (page 13) and `CFSocketDisableCallBacks` (page 12).

The `kCFSocketReadCallBack`, `kCFSocketAcceptCallBack`, and `kCFSocketDataCallBack` callbacks are mutually exclusive.

Version Notes

`kCFSocketWriteCallBack` is available in Mac OS X v10.2 and later.

CFSocket Flags

Flags that can be set on a CFSocket object to control its behavior.

```
enum {
    kCFSocketAutomaticallyReenableReadCallBack = 1,
    kCFSocketAutomaticallyReenableAcceptCallBack = 2,
    kCFSocketAutomaticallyReenableDataCallBack = 3,
    kCFSocketAutomaticallyReenableWriteCallBack = 8,
    kCFSocketCloseOnInvalidate = 128
};
```

Constants

`kCFSocketAutomaticallyReenableReadCallBack`

When enabled using `CFSocketSetSocketFlags` (page 18), the read callback is called every time the sockets has data to be read. When disabled, the read callback is called only once the next time data are available. The read callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableAcceptCallBack`

When enabled using `CFSocketSetSocketFlags` (page 18), the accept callback is called every time someone connects to your socket. When disabled, the accept callback is called only once the next time a new socket connection is accepted. The accept callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableDataCallBack`

When enabled using `CFSocketSetSocketFlags` (page 18), the data callback is called every time the socket has read some data. When disabled, the data callback is called only once the next time data are read. The data callback is automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketAutomaticallyReenableWriteCallBack`

When enabled using `CFSocketSetSocketFlags` (page 18), the write callback is called every time more data can be written to the socket. When disabled, the write callback is called only the next time data can be written. The write callback is not automatically reenabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

`kCFSocketCloseOnInvalidate`

When enabled using `CFSocketSetSocketFlags` (page 18), the native socket associated with a CFSocket object is closed when the CFSocket object is invalidated. When disabled, the native socket remains open. This option is enabled by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocket.h`.

Discussion

The flags for a CFSocket object are set with `CFSocketSetSocketFlags` (page 18). To immediately enable or disable a callback, use `CFSocketEnableCallBacks` (page 13) and `CFSocketDisableCallBacks` (page 12).

Error Codes

Error codes for many CFSocket functions.

```
enum CFSocketError {
    kCFSocketSuccess = 0,
    kCFSocketError = -1,
    kCFSocketTimeout = -2
};
typedef enum CFSocketError CFSocketError;
```

Constants

`kCFSocketSuccess`

The socket operation succeeded.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketError`

The socket operation failed.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

`kCFSocketTimeout`

The socket operation timed out.

Available in Mac OS X v10.0 and later.

Declared in `CFSocket.h`.

Document Revision History

This table describes the changes to *CFSocket Reference*.

Date	Notes
2008-10-15	Added links to companion guides.
2006-07-06	Made minor formatting changes.
2006-06-28	Updated information regarding INVALID_SOCKET.
2006-04-04	Replaced the term "connection rendezvous sockets" with "listening sockets."
2006-02-07	Made formatting changes.
2005-11-09	Removed reference to retired document.
2005-08-11	Fixed typo in discussion of callback types.
2003-01-01	First version of this document.

Index