CFNetwork Framework Reference

Networking & Internet

2010-02-24

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

Simultaneously published in the United States and Canada.

Contents

Introduction

Framework:	CFNetwork/CFNetwork.h
Declared in	CFFTPStream.h CFHTTPAuthentication.h CFHTTPMessage.h CFHTTPStream.h CFHost.h CFNetDiagnostics.h CFNetServices.h CFNetworkErrors.h CFProxySupport.h CFSocketStream.h CFStream.h

This collection of documents provides the API reference for the CFNetwork framework.

Introduction

Opaque Types

PART I

Opaque Types

CFFTPStream Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFFTPStream.h
Companion guide	CFNetwork Programming Guide

Overview

This document describes the CFStream functions for working with FTP connections. It is part of the CFFTP API.

Functions

CFFTPCreateParsedResourceListing

Parses an FTP listing to a dictionary.

```
CFIndex CFFTPCreateParsedResourceListing (
   CFAllocatorRef alloc,
   const UInt8 *buffer,
   CFIndex bufferLength,
   CFDictionaryRef *parsed
);
```

Parameters

alloc

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

buffer

A pointer to a buffer holding zero or more lines of resource listing.

bufferLength

The length in bytes of the buffer pointed to by `buffer`.

parsed

Upon return, contains a dictionary containing the parsed resource information. If parsing fails, a `NULL` pointer is returned.

Return Value

The number of bytes parsed, `0` if no bytes were available for parsing, or `-1` if parsing failed.

Discussion
This function examines the contents of buffer as an FTP directory listing and parses into a CFDictionary the information for a single file or folder. The CFDictionary is returned in the `parsed` parameter, and the number of bytes used from buffer is returned.

Availability
Available in iOS 2.0 and later.

Declared In
`CFFTPStream.h`

CFReadStreamCreateWithFTPURL

Creates an FTP read stream.

```
CFReadStreamRef CFReadStreamCreateWithFTPURL (
   CFAllocatorRef alloc,
   CFURLRef ftpURL
);
```

Parameters

`alloc`

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

`ftpURL`

A pointer to a CFURL structure for the URL to be downloaded that can be created by calling any of the `CFURLCreate` functions, such as `CFURLCreateWithString`.

Return Value
A new read stream, or `NULL` if the call failed. Ownership follows the Create Rule.

Discussion
This function creates an FTP read stream for downloading data from an FTP URL. If the `ftpURL` parameter is created with the user name and password as part of the URL (such as `ftp://username:password@ftp.example.com`) then the user name and password will automatically be set in the `CFReadStream`. Otherwise, call `CFReadStreamSetProperty` to set the steam's properties, such as `kCFStreamPropertyFTPUserName` and `kCFStreamPropertyFTPPassword` to associate a user name and password with the stream that are used to log in when the stream is opened. See "Constants" (page 11) for a description of all FTP stream properties.

To initiate a connection with the FTP server, call `CFReadStreamOpen`. To read the FTP stream, call `CFReadStreamRead`. If the URL refers to a directory, the stream provides the listing results sent by the server. If the URL refers to a file, the stream provides the data in that file.

To close a connection with the FTP server, call `CFReadStreamClose`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFFTPStream.h`

CFWriteStreamCreateWithFTPURL

Creates an FTP write stream.

```
CFWriteStreamRef CFWriteStreamCreateWithFTPURL (
   CFAllocatorRef alloc,
   CFURLRef ftpURL
);
```

Parameters

alloc

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

ftpURL

A pointer to a CFURL structure for the URL to be uploaded created by calling any of the `CFURLCreate` functions, such as `CFURLCreateWithString`.

Return Value
A new write stream, or `NULL` if the call failed. Ownership follows the Create Rule.

Discussion
This function creates an FTP write stream for uploading data to an FTP URL. If the `ftpURL` parameter is created with the user name and password as part of the URL (such as `ftp://username:password@ftp.example.com`) then the user name and password will automatically be set in the `CFWriteStream`. Call `CFWriteStreamSetProperty` to set the steam's properties, such as `kCFStreamPropertyFTPUserName` and `kCFStreamPropertyFTPPassword` to associate a user name and password with the stream that are used to log in when the stream is opened. See "Constants" (page 11) for a description of all FTP stream properties.

After creating the write stream, you can call `CFWriteStreamGetStatus` at any time to check the status of the stream.

To initiate a connection with the FTP server, call `CFWriteStreamOpen`. If the URL specifies a directory, the open is immediately followed by the event `kCFStreamEventEndEncountered` (and the stream passes to the state `kCFStreamStatusAtEnd`). Once the stream reaches this state, the directory has been created. Intermediary directories are not created.

To write to the FTP stream, call `CFWriteStreamWrite`.

To close a connection with the FTP server, call `CFWriteStreamClose`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFFTPStream.h`

Constants

CFStream FTP Property Constants

Constants for setting and copying CFStream FTP properties.

```
const CFStringRef kCFStreamPropertyFTPUserName;
const CFStringRef kCFStreamPropertyFTPPassword;
const CFStringRef kCFStreamPropertyFTPUsePassiveMode;
const CFStringRef kCFStreamPropertyFTPResourceSize;
const CFStringRef kCFStreamPropertyFTPFetchResourceInfo;
const CFStringRef kCFStreamPropertyFTPFileTransferOffset;
const CFStringRef kCFStreamPropertyFTPAttemptPersistentConnection;
const CFStringRef kCFStreamPropertyFTPProxy;
const CFStringRef kCFStreamPropertyFTPProxyHost;
extern const CFStringRef kCFStreamPropertyFTPProxyPort;
extern const CFStringRef kCFStreamPropertyFTPProxyUser;
extern const CFStringRef kCFStreamPropertyFTPProxyPassword;
```

Constants

`kCFStreamPropertyFTPUserName`

FTP User Name stream property key for set and copy operations. A value of type CFString for storing the login user name. Don't set this property when anonymous FTP is desired.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPPassword`

FTP Password stream property key for set and copy operations. A value of type CFString for storing the login password. Don't set this property when anonymous FTP is desired.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPUsePassiveMode`

FTP Passive Mode stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to enable passive mode; set this property to `kCFBooleanFalse` to disable passive mode.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPResourceSize`

FTP Resource Size read stream property key copy operations. This property stores a CFNumber of type `kCFNumberLongLongType` representing the size of a resource in bytes.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPFetchResourceInfo`

FTP Fetch Resource Information stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to require that resource information, such as size, must be provided before download starts; set this property to `kCFBooleanFalse` to allow downloads to start without resource information. For this version, size is the only resource information.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPFileTransferOffset`

FTP File Transfer Offset stream property key for set and copy operations. The value of this property is a CFNumber of type `kCFNumberLongLongType` representing the file offset at which to start the transfer.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPAttemptPersistentConnection`
: FTP Attempt Persistent Connection stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to enable the reuse of existing server connections; set this property to `kCFBooleanFalse` to not reuse existing server connections. By default, this property is set to `kCFBooleanTrue`.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxy`
: FTP Proxy stream property key for set and copy operations. The property is a value of type CFDictionary that holds proxy dictionary key-value pairs. The dictionary returned by SystemConfiguration can also be set as the value of this property.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyHost`
: FTP Proxy Host stream property key or an FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the host name of a proxy server. This property can be set and copied individually or via a CFDictionary. This property is the same as the `kSCPropNetProxiesFTPProxy` property defined in `SCSchemaDefinitions.h`.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyPort`
: FTP Proxy Port stream property key or an FTP Proxy dictionary key for set and copy operations. The value of this property is a CFNumber of type `kCFNumberIntType` containing the port number of a proxy server. This property can be set and copied individually or via a CFDictionary. This property is the same as the `kSCPropNetProxiesFTPPort` property defined in `SCSchemaDefinitions.h`.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyUser`
: FTP Proxy Host stream property key or FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the username to be used when connecting to the proxy server.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyPassword`
: FTP Proxy Port stream property key or FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the password to be used when connecting to the proxy server.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

Discussion
The CFStream property constants are used to specify the property to set when calling `CFReadStreamSetProperty` or `CFWriteStreamSetProperty` and to copy when calling `CFReadStreamCopyProperty` or `CFWriteStreamCopyProperty`. They can also be passed to a CFDictionary creator or to an item accessor or mutator.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFNetwork/CFFTPStream.h`

CFStream FTP Resource Constants

FTP resource constants.

```
const CFStringRef kCFFTPResourceMode;
const CFStringRef kCFFTPResourceName;
const CFStringRef kCFFTPResourceOwner;
const CFStringRef kCFFTPResourceGroup;
const CFStringRef kCFFTPResourceLink;
const CFStringRef kCFFTPResourceSize;
const CFStringRef kCFFTPResourceType;
const CFStringRef kCFFTPResourceModDate;
```

Constants

`kCFFTPResourceMode`
CFDictionary key for getting the CFNumber containing the access permissions, defined in `sys/types.h`, of the FTP resource.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceName`
CFDictionary key for getting the CFString containing the name of the FTP resource.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceOwner`
CFDictionary key for getting the CFString containing the name of the owner of the FTP resource.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceGroup`
CFDictionary key for getting the CFString containing the name of a group that shares the FTP resource.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceLink`
CFDictionary key for getting the CFString containing the symbolic link information. If the item is a symbolic link, the CFString contains the path to the item that the link references.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceSize`
CFDictionary key for getting the CFNumber containing the size in bytes of the FTP resource.

Available in iOS 2.0 and later.

Declared in `CFFTPStream.h`.

`kCFFTPResourceType`
: CFDictionary key for getting the CFNumber containing the type of the FTP resource as defined in `sys/dirent.h`.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceModDate`
: CFDictionary key for getting the CFDate containing the last date and time the FTP resource was modified.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

Discussion
The values of FTP resource keys are extracted from a line of the directory list by the `CFFTPCreateParsedResourceListing` (page 9) function.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFNetwork/CFFTPStream.h`

Error Domains

Error domains specific to `CFFTPStream` calls.

```
extern const SInt32 kCFStreamErrorDomainFTP;
```

Constants
`kCFStreamErrorDomainFTP`
: Error domain that returns the last result code returned by the FTP server.

  Available in iOS 2.0 and later.

  Declared in `CFFTPStream.h`.

Discussion
To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFHost Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHost.h
Companion guide	CFNetwork Programming Guide

Overview

The CFHost API allows you to create instances of the CFHost object that you can use to acquire host information, including names, addresses, and reachability information.

The process of acquiring information about a host is known as resolution. Begin by calling `CFHostCreateWithAddress` or `CFHostCreateWithName` to create an instance of a CFHost using an address or a name, respectively. If you want to resolve the host asynchronously. call `CFHostSetClient` to associate your client context and user-defined callback function with the host. Then call `CFHostScheduleWithRunLoop` to schedule the host on a run loop.

To start resolution, call `CFHostStartInfoResolution`. If you set up for asynchronous resolution, `CFHostStartInfoResolution` returns immediately. Your callback function will be called when resolution is complete. If you didn't set up for asynchronous resolution, `CFHostStartInfoResolution` blocks until resolution is complete, an error occurs, or the resolution is cancelled.

When resolution is complete, call `CFHostGetAddressing` or `CFHostGetNames` to get an array of known addresses or known names, respectively, for the host. Call `CFHostGetReachability` to get flags, declared in `SystemConfiguration/SCNetwork.h`, that describe the reachability of the host.

When you no longer need a CFHost object, call `CFHostUnscheduleFromRunLoop` and `CFHostSetClient` to disassociate the host from your user-defined client context and callback function (if it was set up for asynchronous resolution). Then dispose of it.

Functions by Task

Creating a host

`CFHostCreateCopy` (page 19)
Creates a new host object by copying.

`CFHostCreateWithAddress` (page 19)
Uses an address to create an instance of a host object.

`CFHostCreateWithName` (page 20)
Uses a name to create an instance of a host object.

CFHost Functions

`CFHostCancelInfoResolution` (page 18)
Cancels the resolution of a host.

`CFHostGetAddressing` (page 21)
Gets the addresses from a host.

`CFHostGetNames` (page 21)
Gets the names from a CFHost.

`CFHostGetReachability` (page 22)
Gets reachability information from a host.

`CFHostStartInfoResolution` (page 24)
Starts resolution for a host object.

`CFHostSetClient` (page 24)
Associates a client context and a callback function with a CFHost object or disassociates a client context and callback function that were previously set.

`CFHostScheduleWithRunLoop` (page 23)
Schedules a CFHost on a run loop.

`CFHostUnscheduleFromRunLoop` (page 25)
Unschedules a CFHost from a run loop.

Getting the CFHost Type ID

`CFHostGetTypeID` (page 23)
Gets the Core Foundation type identifier for the CFHost opaque type.

Functions

CFHostCancelInfoResolution

Cancels the resolution of a host.

```
void CFHostCancelInfoResolution (
   CFHostRef theHost,
   CFHostInfoType info
);
```

Parameters

theHost
The host for which a resolution is to be cancelled. This value must not be `NULL`.

info

A value of type `CFHostInfoType` specifying the type of resolution that is to be cancelled. See CFHostInfoType Constants (page 28) for possible values.

Discussion
This function cancels the asynchronous or synchronous resolution specified by `info` for the host specified by `theHost`.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostCreateCopy

Creates a new host object by copying.

```
CFHostRef CFHostCreateCopy (
   CFAllocatorRef alloc,
   CFHostRef host
);
```

Parameters

alloc

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

addr

The host to copy. This value must not be `NULL`.

Return Value
A valid CFHost object or `NULL` if the copy could not be created. The new host contains a copy of all previously resolved data from the original host. Ownership follows the Create Rule.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostCreateWithAddress

Uses an address to create an instance of a host object.

```
CFHostRef CFHostCreateWithAddress (
   CFAllocatorRef allocator,
   CFDataRef addr
);
```

Parameters

alloc

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

addr

A CFDataRef object containing a `sockaddr` structure for the address of the host. This value must not be `NULL`.

Return Value

A valid CFHostRef object that can be resolved, or `NULL` if the host could not be created. Ownership follows the Create Rule.

Discussion

Call `CFHostStartInfoResolution` (page 24) to resolve the return object's name and reachability information.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHost.h`

CFHostCreateWithName

Uses a name to create an instance of a host object.

```
CFHostRef CFHostCreateWithName (
   CFAllocatorRef allocator,
   CFStringRef hostname
);
```

Parameters

alloc

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

hostname

A string representing the name of the host. This value must not be `NULL`.

Return Value

A valid CFHostRef object that can be resolved, or `NULL` if the host could not be created. Ownership follows the Create Rule.

Discussion

Call `CFHostStartInfoResolution` (page 24) to resolve the object's addresses and reachability information.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostGetAddressing

Gets the addresses from a host.

```
CFArrayRef CFHostGetAddressing (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

`theHost`

The CFHost whose addresses are to be obtained. This value must not be `NULL`.

`hasBeenResolved`

On return, a pointer to a Boolean that is `TRUE` if addresses were available and `FALSE` if addresses were not available. This parameter can be null.

`function result`

A CFArray of addresses where address is a `sockaddr` structure wrapped by a CFDataRef, or null if no addresses were available.

Discussion
This function gets the addresses from a CFHost. The CFHost must have been previously resolved. To resolve a CFHost, call `CFHostStartInfoResolution` (page 24).

Special Considerations

This function gets the addresses in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostGetNames

Gets the names from a CFHost.

```
CFArrayRef CFHostGetNames (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

theHost

The host to examine. The host must have been previously resolved. (To resolve a host, call `CFHostStartInfoResolution` (page 24).) This value must not be `NULL`.

hasBeenResolved

On return, contains `TRUE` if names were available, otherwise `FALSE`. This value may be `NULL`.

Return Value

An array containing the of names of *theHost*, or `NULL` if no names were available.

Special Considerations

This function gets the names in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHost.h`

CFHostGetReachability

Gets reachability information from a host.

```
CFDataRef CFHostGetReachability (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

theHost

The host whose reachability is to be obtained. The host must have been previously resolved. (To resolve a host, call `CFHostStartInfoResolution` (page 24).) This value must not be `NULL`.

hasBeenResolved

On return, contains `TRUE` if the reachability was available, otherwise `FALSE`. This value may be `NULL`.

Return Value

A CFData object that wraps the reachability flags (`SCNetworkConnectionFlags`) defined in `SystemConfiguration/SCNetwork.h`, or `NULL` if reachability information was not available.

Special Considerations

This function gets reachability information in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostGetTypeID

Gets the Core Foundation type identifier for the CFHost opaque type.

```
CFTypeID CFHostGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHost opaque type.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostScheduleWithRunLoop

Schedules a CFHost on a run loop.

```
void CFHostScheduleWithRunLoop (
   CFHostRef theHost,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theHost
The host to be schedule on a run loop. This value must not be `NULL`.

runLoop
The run loop on which to schedule *theHost*. This value must not be `NULL`.

runLoopMode
The mode on which to schedule *theHost*. This value must not be `NULL`.

Discussion
Schedules *theHost* on a run loop, which causes resolutions of the host to be performed asynchronously. The caller is responsible for ensuring that at least one of the run loops on which the host is scheduled is being run.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostSetClient

Associates a client context and a callback function with a CFHost object or disassociates a client context and callback function that were previously set.

```
Boolean CFHostSetClient (
   CFHostRef theHost,
   CFHostClientCallBack clientCB,
   CFHostClientContext *clientContext
);
```

Parameters

theHost

The host to modify. The value must not be `NULL`.

clientCB

The callback function to associate with *theHost*. The callback function will be called when a resolution completes or is cancelled. If you are calling this function to disassociate a client context and callback from `theHost`, p`clientCB`ass `NULL`.

clientContext

A `CFHostClientContext` (page 27) structure whose `info` field will be passed to the callback function specified by `clientCB` when `clientCB` is called. This value must not be `NULL` when setting an association.

Pass `NULL` when disassociating a client context and a callback from a host.

Return Value

`TRUE` if the association could be set or unset, otherwise `FALSE`.

Discussion

The callback function specified by `clientCB` will be called when a resolution completes or is cancelled.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHost.h`

CFHostStartInfoResolution

Starts resolution for a host object.

```
Boolean CFHostStartInfoResolution (
   CFHostRef theHost,
   CFHostInfoType info,
   CFStreamError *error
);
```

Parameters

theHost

The host, obtained by previously calling `CFHostCreateCopy` (page 19), `CFHostCreateWithAddress` (page 19), or `CFHostCreateWithName` (page 20), that is to be resolved. This value must not be `NULL`.

info

A value of type `CFHostInfoType` specifying the type of information that is to be retrieved. See CFHostInfoType Constants (page 28) for possible values.

error

A pointer to a `CFStreamError` structure, that, if an error occurs, is set to the error and the error's domain. In synchronous mode, the error indicates why resolution failed, and in asynchronous mode, the error indicates why resolution failed to start.

Return Value
`TRUE` if the resolution was started (asynchronous mode); `FALSE` if another resolution is already in progress for *theHost* or if an error occurred.

Discussion
This function retrieves the information specified by `info` and stores it in the host.

In synchronous mode, this function blocks until the resolution has completed, in which case this function returns `TRUE`, until the resolution is stopped by calling `CFHostCancelInfoResolution` (page 18) from another thread, in which case this function returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostUnscheduleFromRunLoop

Unschedules a CFHost from a run loop.

```
void CFHostUnscheduleFromRunLoop (
   CFHostRef theHost,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService

The host to unschedule. This value must not be `NULL`.

runLoop

The run loop. This value must not be `NULL`.

runLoopMode

The mode from which the service is to be unscheduled. This value must not be `NULL`.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

Callbacks

CFHostClientCallBack

Defines a pointer to the callback function that is called when an asynchronous resolution of a CFHost completes or an error occurs for an asynchronous CFHost resolution.

```
typedef void (CFHostClientCallBack) (
   CFHostRef theHost,
   CFHostInfoType typeInfo,
   const CFStreamError *error,
   void *info);
```

If you name your callback `MyHostClientCallBack`, you would declare it like this:

```
void MyHostClientCallBack (
   CFHostRef theHost,
   CFHostInfoType typeInfo,
   const CFStreamError *error,
   void *info
);
```

Parameters

`theHost`
The host for which an asynchronous resolution has been completed.

`typeInfo`
Value of type `CFHostInfoType` representing the type of information (addresses, names, or reachability information) obtained by the completed resolution. See CFHostInfoType Constants (page 28) for possible values.

`error`
If the resolution failed, contains a `CFStreamError` structure whose `error` field contains an error code.

`info`
User-defined context information. The value pointed to by `info` is the same as the value pointed to by the `info` field of the `CFHostClientContext` (page 27) structure that was provided when the host was associated with this callback function.

Discussion
The callback function for a CFHost object is called one or more times when an asynchronous resolution completes for the specified host, when an asynchronous resolution is cancelled, or when an error occurs during an asynchronous resolution.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

Data Types

CFHostRef

An opaque reference representing an CFHost object.

```
typedef struct __CFHost* CFHostRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

CFHostClientContext

A structure containing user-defined data and callbacks for CFHost objects.

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

Fields

`version`
: The version number of the structure type passed as a parameter to the host client function. The only valid version number is `0`.

`info`
: An arbitrary pointer to allocated memory containing user-defined data that can be associated with the host and that is passed to the callbacks.

`retain`
: The callback used to add a retain for the host on the info pointer for the life of the host, and may be used for temporary references the host needs to take. This callback returns the actual info pointer to store in the host, almost always just the pointer passed as the parameter.

`release`
: The callback used to remove a retain previously added for the host on the info pointer.

`copyDescription`
: The callback used to create a descriptive string representation of the info pointer (or the data pointed to by the info pointer) for debugging purposes. This callback is called by the `CFCopyDescription` function.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHost.h`

Constants

CFHostInfoType Constants

Values indicating the type of data that is to be resolved or the type of data that was resolved.

```
enum CFHostInfoType {
    kCFHostAddresses = 0,
    kCFHostNames = 1,
    kCFHostReachability = 2
};
typedef enum CFHostInfoType CFHostInfoType;
```

Constants

`kCFHostAddresses`

Specifies that addresses are to be resolved or that addresses were resolved.

Available in iOS 2.0 and later.

Declared in `CFHost.h`.

`kCFHostNames`

Specifies that names are to be resolved or that names were resolved.

Available in iOS 2.0 and later.

Declared in `CFHost.h`.

`kCFHostReachability`

Specifies that reachability information is to be resolved or that reachability information was resolved.

Available in iOS 2.0 and later.

Declared in `CFHost.h`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFNetwork/CFHost.h`

Error Domains

Error domains specific to `CFHost` calls.

```
extern const SInt32 kCFStreamErrorDomainNetDB;
extern const SInt32 kCFStreamErrorDomainSystemConfiguration;
```

Constants

`kCFStreamErrorDomainNetDB`

The error domain that returns errors from the network database (DNS resolver) layer (described in `/usr/include/netdb.h`).

Available in iOS 2.0 and later.

Declared in `CFHost.h`.

`kCFStreamErrorDomainSystemConfiguration`

The error domain that returns errors from the system configuration layer (described in *System Configuration Framework Reference*).

Available in iOS 2.0 and later.

Declared in `CFHost.h`.

Discussion

To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFHTTPAuthentication Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHTTPAuthentication.h
Companion guide	CFNetwork Programming Guide

Overview

The CFHTTPAuthentication opaque type provides an abstraction of HTTP authentication information.

Functions by Task

Creating an HTTP authentication

`CFHTTPAuthenticationCreateFromResponse` (page 34)
Uses an authentication failure response to create a CFHTTPAuthentication object.

CFHTTP Authentication Functions

This section describes the CFNetwork authentication functions that are used to manage authentication information associated with a request. The functions work with a CFHTTPAuthentication object, which is created from an HTTP response that failed with a 401 or 407 error code.

When you have analyzed the CFHTTPAuthentication object and acquired the necessary credentials to perform the authentication, call `CFHTTPMessageApplyCredentials` (page 45) or `CFHTTPMessageApplyCredentialDictionary` (page 44) to perform the authentication.

`CFHTTPAuthenticationAppliesToRequest` (page 32)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object is associated with a CFHTTPMessage object.

`CFHTTPAuthenticationCopyDomains` (page 33)
Returns an array of domain URLs to which a given CFHTTPAuthentication object can be applied.

`CFHTTPAuthenticationCopyMethod` (page 33)
Gets the strongest authentication method that will be used when a CFHTTPAuthentication object is applied to a request.

`CFHTTPAuthenticationCopyRealm` (page 33)
Gets an authentication information's namespace.

`CFHTTPAuthenticationIsValid` (page 35)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object is valid.

`CFHTTPAuthenticationRequiresAccountDomain` (page 36)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires an account domain.

`CFHTTPAuthenticationRequiresOrderedRequests` (page 36)
Returns a Boolean value that indicates whether authentication requests should be made one at a time.

`CFHTTPAuthenticationRequiresUserNameAndPassword` (page 37)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires a username and a password.

Getting the CFHTTPAuthentication type ID

`CFHTTPAuthenticationGetTypeID` (page 35)
Gets the Core Foundation type identifier for the CFHTTPAuthentication opaque type.

Functions

CFHTTPAuthenticationAppliesToRequest

Returns a Boolean value that indicates whether a CFHTTPAuthentication object is associated with a CFHTTPMessage object.

```
Boolean CFHTTPAuthenticationAppliesToRequest (
   CFHTTPAuthenticationRef auth,
   CFHTTPMessageRef request
);
```

Parameters

auth
The CFHTTPAuthentication object to examine.

request
Request that `auth` is to be tested against.

Return Value
`TRUE` if `auth` is associated with `request`, otherwise `FALSE`.

Discussion
If this function returns `TRUE`, you can use *auth* to provide authentication information when using *request*.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyDomains

Returns an array of domain URLs to which a given CFHTTPAuthentication object can be applied.

```
CFArrayRef CFHTTPAuthenticationCopyDomains (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value

A CFArray object that contains the domain URL's to which `auth` should be applied. Ownership follows the Create Rule.

Discussion

This function is provided for informational purposes only.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyMethod

Gets the strongest authentication method that will be used when a CFHTTPAuthentication object is applied to a request.

```
CFStringRef CFHTTPAuthenticationCopyMethod (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value

A string containing the authentication method that will be used *auth* is applied to a request. If more than one authentication method is available, the strongest authentication method is returned. Ownership follows the Create Rule.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyRealm

Gets an authentication information's namespace.

```
CFStringRef CFHTTPAuthenticationCopyRealm (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value
The namespace, if there is one; otherwise `NULL`. Ownership follows the Create Rule.

Discussion
Some authentication methods provide a namespace, and it is usually used to prompt the user for a name and password.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationCreateFromResponse

Uses an authentication failure response to create a CFHTTPAuthentication object.

```
CFHTTPAuthenticationRef CFHTTPAuthenticationCreateFromResponse (
   CFAllocatorRef alloc,
   CFHTTPMessageRef response
);
```

Parameters

alloc

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

response

Response indicating an authentication failure; usually a 401 or a 407 response.

Return Value
CFHTTPAuthentication object that can be used for adding credentials to future requests. Ownership follows the Create Rule.

Discussion
This function uses a response containing authentication failure information to create a reference to a CFHTTPAuthentication object. You can use the object to add credentials to future requests. You can query the object to get the following information:

- whether it can be used and re-used to authenticate with its corresponding server [`CFHTTPAuthenticationIsValid` (page 35)]
- the authentication method that will be used when it is used to perform an authentication [`CFHTTPAuthenticationCopyMethod` (page 33)]
- whether it is associated with a particular CFHTTPMessageRef [`CFHTTPAuthenticationAppliesToRequest` (page 32)

- whether a user name and a password will be required when it is used to perform an authentication [CFHTTPAuthenticationRequiresUserNameAndPassword (page 37)]
- whether an account domain will be required when it is used to perform an authentication [CFHTTPAuthenticationRequiresAccountDomain (page 36)]
- whether authentication requests should be sent one at a time to the corresponding server [CFHTTPAuthenticationRequiresOrderedRequests (page 36)]
- the namespace (if any) that the domain uses to prompt for a name and password [CFHTTPAuthenticationCopyRealm (page 33)]
- the domain URLs the instance applies to [CFHTTPAuthenticationCopyDomains (page 33)]

When you have determined what information will be needed to perform the authentication and accumulated that information, call CFHTTPMessageApplyCredentials (page 45) or CFHTTPMessageApplyCredentialDictionary (page 44) to perform the authentication.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPAuthentication.h

CFHTTPAuthenticationGetTypeID

Gets the Core Foundation type identifier for the CFHTTPAuthentication opaque type.

```
CFTypeID CFHTTPAuthenticationGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHTTPAuthentication opaque type.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPAuthentication.h

CFHTTPAuthenticationIsValid

Returns a Boolean value that indicates whether a CFHTTPAuthentication object is valid.

```
Boolean CFHTTPAuthenticationIsValid (
   CFHTTPAuthenticationRef auth,
   CFStreamError *error
);
```

Parameters

auth
The CFHTTPAuthentication object to examine.

error
Pointer to a CFStreamError structure, whose fields, if an error has occurred, are set to the error and the error's domain.

Return Value
`TRUE` if `auth` contains enough information to be applied to a request.

If this function returns `FALSE`, the CFHTTPAuthentication object may still contain useful information, such as the name of an unsupported authentication method.

Discussion
If this function returns `TRUE` for *`auth`*, the object is good for use with functions such as `CFHTTPMessageApplyCredentials` (page 45) and `CFHTTPMessageApplyCredentialDictionary` (page 44). If this function returns `FALSE`, *`auth`* is invalid, and authentications using it will not succeed.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresAccountDomain

Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires an account domain.

```
Boolean CFHTTPAuthenticationRequiresAccountDomain (
   CFHTTPAuthenticationRef auth
);
```

Parameters
`auth`
The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` uses an authentication method that requires an account domain, otherwise `FALSE`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresOrderedRequests

Returns a Boolean value that indicates whether authentication requests should be made one at a time.

```
Boolean CFHTTPAuthenticationRequiresOrderedRequests (
   CFHTTPAuthenticationRef auth
);
```

Parameters
`auth`
The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` requires ordered requests, otherwise `FALSE`.

Discussion
Some authentication methods require that future requests must be performed in an ordered manner. If this function returns `TRUE`, clients can improve their chances of authenticating successfully by issuing requests one at a time as responses come back from the server.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresUserNameAndPassword

Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires a username and a password.

```
Boolean CFHTTPAuthenticationRequiresUserNameAndPassword (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` requires a username and password when it is applied to a request; otherwise, `FALSE`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

Data Types

CFHTTPAuthenticationRef

An opaque reference representing HTTP authentication information.

```
typedef struct __CFHTTPAuthentication *CFHTTPAuthenticationRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPAuthentication.h`

Constants

CFHTTP Authentication Scheme Constants

Specifies the authentication scheme when adding authentication information to a CFHTTP request message object.

```
const CFStringRef kCFHTTPAuthenticationSchemeBasic;
const CFStringRef kCFHTTPAuthenticationSchemeDigest;
const CFStringRef kCFHTTPAuthenticationSchemeNegotiate;
const CFStringRef kCFHTTPAuthenticationSchemeNTLM;
```

Constants

`kCFHTTPAuthenticationSchemeBasic`
Specifies basic authentication consisting of a user name and a password.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeDigest`
Reserved.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNegotiate`
Specifies the Negotiate authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNTLM`
Specifies the NTLM authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

Discussion

The authentication scheme constants are used to specify the authentication scheme when calling `CFHTTPMessageAddAuthentication` (page 43).

CFStream HTTP Authentication Error Constants

Authentication error codes that may be returned when trying to apply authentication to a request.

```
enum CFStreamErrorHTTPAuthentication{
    kCFStreamErrorHTTPAuthenticationTypeUnsupported = -1000,
    kCFStreamErrorHTTPAuthenticationBadUserName = -1001,
    kCFStreamErrorHTTPAuthenticationBadPassword = -1002
};
typedef enum CFStreamErrorHTTPAuthentication CFStreamErrorHTTPAuthentication;
```

Constants

`kCFStreamErrorHTTPAuthenticationTypeUnsupported`
Specified authentication type is not supported.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFStreamErrorHTTPAuthenticationBadUserName`
User name is in a format that is not suitable for the request. Currently, user names are decoded using `kCFStringEncodingISOLatin1`.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFStreamErrorHTTPAuthenticationBadPassword`
Password is in a format that is not suitable for the request. Currently, passwords are decoded using `kCFStringEncodingISOLatin1`.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

CFHTTPMessageApplyCredentialDictionary Keys

Constants for keys in the dictionary passed to `CFHTTPMessageApplyCredentialDictionary` (page 44).

```
const CFStringRef kCFHTTPAuthenticationUsername;
const CFStringRef kCFHTTPAuthenticationPassword;
const CFStringRef kCFHTTPAuthenticationAccountDomain;
```

Constants

`kCFHTTPAuthenticationUsername`
Username to use for authentication.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFHTTPAuthenticationPassword`
Password to use for authentication.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFHTTPAuthenticationAccountDomain`
Account domain to use for authentication.

Available in iOS 2.0 and later.

Declared in `CFHTTPAuthentication.h`.

CFHTTPMessage Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHTTPMessage.h
Companion guides	Getting Started With Networking CFNetwork Programming Guide

Overview

The `CFHTTPMessage` opaque type represents an HTTP message.

Functions by Task

Creating a Message

`CFHTTPMessageCreateCopy` (page 50)
Gets a copy of a CFHTTPMessage object.

`CFHTTPMessageCreateEmpty` (page 50)
Creates and returns a new, empty `CFHTTPMessage` object.

`CFHTTPMessageCreateRequest` (page 51)
Creates and returns a `CFHTTPMessage` object for an HTTP request.

`CFHTTPMessageCreateResponse` (page 52)
Creates and returns a `CFHTTPMessage` object for an HTTP response.

Modifying a message

`CFHTTPMessageAppendBytes` (page 44)
Appends data to a `CFHTTPMessage` object.

`CFHTTPMessageSetBody` (page 54)
Sets the body of a `CFHTTPMessage` object.

`CFHTTPMessageSetHeaderFieldValue` (page 54)
Sets the value of a header field in an HTTP message.

Getting information from a message

`CFHTTPMessageCopyBody` (page 47)
Gets the body from a `CFHTTPMessage` object.

`CFHTTPMessageCopyAllHeaderFields` (page 46)
Gets all header fields from a `CFHTTPMessage` object.

`CFHTTPMessageCopyHeaderFieldValue` (page 47)
Gets the value of a header field from a `CFHTTPMessage` object.

`CFHTTPMessageCopyRequestMethod` (page 48)
Gets the request method from a `CFHTTPMessage` object.

`CFHTTPMessageCopyRequestURL` (page 48)
Gets the URL from a `CFHTTPMessage` object.

`CFHTTPMessageCopySerializedMessage` (page 49)
Serializes a CFHTTPMessage object.

`CFHTTPMessageCopyVersion` (page 49)
Gets the HTTP version from a `CFHTTPMessage` object.

`CFHTTPMessageIsRequest` (page 54)
Returns a boolean indicating whether the CFHTTPMessage is a request or a response.

`CFHTTPMessageIsHeaderComplete` (page 53)
Determines whether a message header is complete.

`CFHTTPMessageGetResponseStatusCode` (page 53)
Gets the status code from a `CFHTTPMessage` object representing an HTTP response.

`CFHTTPMessageCopyResponseStatusLine` (page 49)
Gets the status line from a `CFHTTPMessage` object.

Message authentication

`CFHTTPMessageApplyCredentials` (page 45)
Performs the authentication method specified by a `CFHTTPAuthentication` object.

`CFHTTPMessageApplyCredentialDictionary` (page 44)
Use a dictionary containing authentication credentials to perform the authentication method specified by a `CFHTTPAuthentication` object.

`CFHTTPMessageAddAuthentication` (page 43)
Adds authentication information to a request.

Getting the CFHTTPMessage type identifier

`CFHTTPMessageGetTypeID` (page 53)
Returns the Core Foundation type identifier for the `CFHTTPMessage` opaque type.

Functions

CFHTTPMessageAddAuthentication

Adds authentication information to a request.

```
Boolean CFHTTPMessageAddAuthentication (
   CFHTTPMessageRef request,
   CFHTTPMessageRef authenticationFailureResponse,
   CFStringRef username,
   CFStringRef password,
   CFStringRef authenticationScheme,
   Boolean forProxy
);
```

Parameters

request
The message to which to add authentication information.

authenticationFailureResponse
The response message that contains authentication failure information.

username
The username to add to the request.

password
The password to add to the request.

authenticationScheme
The authentication scheme to use (`kCFHTTPAuthenticationSchemeBasic`, `kCFHTTPAuthenticationSchemeNegotiate`, `kCFHTTPAuthenticationSchemeNTLM`, or `kCFHTTPAuthenticationSchemeDigest`), or pass `NULL` to use the strongest supported authentication scheme provided in the `authenticationFailureResponse` parameter.

forProxy
A flag indicating whether the authentication data that is being added is for a proxy's use (`TRUE`) or for a remote server's use (`FALSE`). If the error code provided by the `authenticationFailureResponse` parameter is 407, set `forProxy` to `TRUE`. If the error code is 401, set `forProxy` to `FALSE`.

Return Value

`TRUE` if the authentication information was successfully added, otherwise `FALSE`.

Discussion

This function adds the authentication information specified by the `username`, `password`, `authenticationScheme`, and `forProxy` parameters to the specified request message. The message referred to by the `authenticationFailureResponse` parameter typically contains a 401 or a 407 error code.

This function is best suited for sending a single request to the server. If you need to send multiple requests, use `CFHTTPMessageApplyCredentials` (page 45).

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageAppendBytes

Appends data to a `CFHTTPMessage` object.

```
Boolean CFHTTPMessageAppendBytes (
   CFHTTPMessageRef message,
   const UInt8 *newBytes,
   CFIndex numBytes
);
```

Parameters

message
The message to modify.

newBytes
A reference to the data to append.

numBytes
The length of the data pointed to by `newBytes`.

Return Value

`TRUE` if the data was successfully appended, otherwise `FALSE`.

Discussion

This function appends the data specified by `newBytes` to the specified message object which was created by calling `CFHTTPMessageCreateEmpty` (page 50). The data is an incoming serialized HTTP request or response received from a client or a server. While appending the data, this function deserializes it, removes any HTTP-based formatting that the message may contain, and stores the message in the message object. You can then call `CFHTTPMessageCopyVersion` (page 49), `CFHTTPMessageCopyBody` (page 47), `CFHTTPMessageCopyHeaderFieldValue` (page 47), and `CFHTTPMessageCopyAllHeaderFields` (page 46) to get the message's HTTP version, the message's body, a specific header field, and all of the message's headers, respectively.

If the message is a request, you can also call `CFHTTPMessageCopyRequestURL` (page 48) and `CFHTTPMessageCopyRequestMethod` (page 48) to get the message's request URL and request method, respectively.

If the message is a response, you can also call `CFHTTPMessageGetResponseStatusCode` (page 53) and `CFHTTPMessageCopyResponseStatusLine` (page 49) to get the message's status code and status line, respectively.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageApplyCredentialDictionary

Use a dictionary containing authentication credentials to perform the authentication method specified by a `CFHTTPAuthentication` object.

```
Boolean CFHTTPMessageApplyCredentialDictionary (
   CFHTTPMessageRef request,
   CFHTTPAuthenticationRef auth,
   CFDictionaryRef dict,
   CFStreamError *error
);
```

Parameters

request

The request for which the authentication method is to be performed.

auth

A `CFHTTPAuthentication` object specifying the authentication method to perform.

dict

A dictionary containing authentication credentials to be applied to the request. For information on the keys in this dictionary, see `CFHTTPAuthenticationRef` (page 37).

error

If an error occurs, upon return contains a `CFStreamError` object that describes the error and the error's domain. Pass `NULL` if you don't want to receive error information.

Return Value

`TRUE` if the authentication was successful, otherwise, `FALSE`.

Discussion

This function performs the authentication method specified by `auth` on behalf of the request specified by `request` using the credentials contained in the dictionary specified by `dict`. The dictionary must contain values for the `kCFHTTPAuthenticationUsername` and `kCFHTTPAuthenticationPassword` keys. If `CFHTTPAuthenticationRequiresAccountDomain` (page 36) returns `TRUE` for *auth*, the dictionary must also contain a value for the `kCFHTTPAuthenticationAccountDomain` key.

Special Considerations

This function is thread safe as long as another thread does not alter the same `CFHTTPAuthentication` object at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPMessageApplyCredentials

Performs the authentication method specified by a `CFHTTPAuthentication` object.

```
Boolean CFHTTPMessageApplyCredentials (
   CFHTTPMessageRef request,
   CFHTTPAuthenticationRef auth,
   CFStringRef username,
   CFStringRef password,
   CFStreamError *error
);
```

Parameters

request

Request for which the authentication method is to be performed.

auth

A `CFHTTPAuthentication` object specifying the authentication method to perform.

username

Username for performing the authentication.

password

Password for performing the authentication.

error

If an error occurs, upon return contains a `CFStreamError` object that describes the error and the error's domain. Pass `NULL` if you don't want to receive error information.

Return Value

`TRUE` if the authentication was successful, otherwise, `FALSE`.

Discussion

This function performs the authentication method specified by `auth` on behalf of the request specified by `request` using the credentials specified by `username` and `password`. If, in addition to a username and password, you also need to specify an account domain, call `CFHTTPMessageApplyCredentialDictionary` (page 44) instead of this function.

This function is appropriate for performing several authentication requests. If you only need to make a single authentication request, consider using `CFHTTPMessageAddAuthentication` (page 43) instead.

Special Considerations

This function is thread safe as long as another thread does not alter the same `CFHTTPMessage` object at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPMessageCopyAllHeaderFields

Gets all header fields from a `CFHTTPMessage` object.

```
CFDictionaryRef CFHTTPMessageCopyAllHeaderFields (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFDictionary` object containing keys and values that are `CFString` objects, where the key is the header fieldname and the dictionary value is the header field's value. Returns `NULL` if the header fields could not be copied. Ownership follows the Create Rule.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCopyBody

Gets the body from a `CFHTTPMessage` object.

```
CFDataRef CFHTTPMessageCopyBody (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFData` object or `NULL` if there was a problem creating the object or if the there is no message body. Ownership follows the Create Rule.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCopyHeaderFieldValue

Gets the value of a header field from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyHeaderFieldValue (
   CFHTTPMessageRef message,
   CFStringRef headerField
);
```

Parameters

message

The message to examine.

headerField
The header field to copy.

Return Value
A `CFString` object containing a copy of the field specified by *headerField*, or `NULL` if there was a problem creating the object of if the specified header does not exist. Ownership follows the Create Rule.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyRequestMethod

Gets the request method from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyRequestMethod (
   CFHTTPMessageRef request
);
```

Parameters
request
The message to examine. This must be a request message.

Return Value
A `CFString` object containing a copy of the message's request method, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyRequestURL

Gets the URL from a `CFHTTPMessage` object.

```
CFURLRef CFHTTPMessageCopyRequestURL (
   CFHTTPMessageRef request
);
```

Parameters
request
The message to examine. This must be a request message.

Return Value
A `CFURLRef` object containing the URL or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyResponseStatusLine

Gets the status line from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyResponseStatusLine (
   CFHTTPMessageRef response
);
```

Parameters

response

The message to examine. This must be a response message.

Return Value
A string containing the message's status line, or `NULL` if there was a problem creating the object. The status line includes the message's protocol version and a success or error code. Ownership follows the Create Rule.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopySerializedMessage

Serializes a CFHTTPMessage object.

```
CFDataRef CFHTTPMessageCopySerializedMessage (
   CFHTTPMessageRef request
);
```

Parameters

request

The message to serialize.

Return Value
A `CFData` object containing the serialized message, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a copy of a `CFHTTPMessage` object in serialized format that is ready for transmission.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyVersion

Gets the HTTP version from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyVersion (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFString` object or `NULL`, if there was a problem creating the object. Ownership follows the Create Rule.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateCopy

Gets a copy of a CFHTTPMessage object.

```
CFHTTPMessageRef CFHTTPMessageCreateCopy (
   CFAllocatorRef alloc,
   CFHTTPMessageRef message
);
```

Parameters

allocator

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

message

The message to copy.

Return Value

A `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

This function returns a copy of a `CFHTTPMessage` object that you can modify, for example, by calling `CFHTTPMessageCopyHeaderFieldValue` (page 47) or by calling`CFHTTPMessageSetBody` (page 54). Then serialize the message by calling`CFHTTPMessageCopySerializedMessage` (page 49) and send the serialized message to a client or a server.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateEmpty

Creates and returns a new, empty `CFHTTPMessage` object.

```
CFHTTPMessageRef CFHTTPMessageCreateEmpty (
   CFAllocatorRef alloc,
   Boolean isRequest
);
```

Parameters

allocator

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

isRequest

A flag that determines whether to create an empty message request or an empty message response. Pass `TRUE` to create an empty request message; pass `FALSE` to create an empty response message.

Return Value

A new `CFHTTPMessage` object or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

Call `CFHTTPMessageAppendBytes` (page 44) to store an incoming, serialized HTTP request or response message in the empty message object.

Availability

Available in iOS 2.0 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateRequest

Creates and returns a `CFHTTPMessage` object for an HTTP request.

```
CFHTTPMessageRef CFHTTPMessageCreateRequest (
   CFAllocatorRef alloc,
   CFStringRef requestMethod,
   CFURLRef url,
   CFStringRef httpVersion
);
```

Parameters

allocator

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

requestMethod

The request method for the request. Use any of the request methods allowed by the HTTP version specified by *httpVersion*.

url

The URL to which the request will be sent.

httpVersion

The HTTP version for this message. Pass `kCFHTTPVersion1_0` or `kCFHTTPVersion1_1`.

Return Value

A new `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a `CFHTTPMessage` object that you can use to build an HTTP request. Continue building the request by calling`CFHTTPMessageSetBody` (page 54) to set the message's body. Call `CFHTTPMessageCopyHeaderFieldValue` (page 47) to set the message's headers.

If you are using a `CFReadStream` object to send the message, call `CFReadStreamCreateForHTTPRequest` (page 57) to create a read stream for the request. If you are not using `CFReadStream`, call `CFHTTPMessageCopySerializedMessage` (page 49) to make the message ready for transmission by serializing it.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCreateResponse

Creates and returns a `CFHTTPMessage` object for an HTTP response.

```
CFHTTPMessageRef CFHTTPMessageCreateResponse (
   CFAllocatorRef alloc,
   CFIndex statusCode,
   CFStringRef statusDescription,
   CFStringRef httpVersion
);
```

Parameters

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

statusCode
The status code for this message response. The status code can be any of the status codes defined in section 6.1.1 of RFC 2616.

statusDescription
The description that corresponds to the status code. Pass NULL to use the standard description for the given status code, as found in RFC 2616.

httpVersion
The HTTP version for this message response. Pass `kCFHTTPVersion1_0` or `kCFHTTPVersion1_1`.

Return Value
A new `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a `CFHTTPMessage` object that you can use to build an HTTP response. Continue building the response by calling`CFHTTPMessageSetBody` (page 54) to set the message's body. Call `CFHTTPMessageSetHeaderFieldValue` (page 54) to set the message's headers. Then call `CFHTTPMessageCopySerializedMessage` (page 49) to make the message ready for transmission by serializing it.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageGetResponseStatusCode

Gets the status code from a CFHTTPMessage object representing an HTTP response.

```
CFIndex CFHTTPMessageGetResponseStatusCode (
   CFHTTPMessageRef response
);
```

Parameters

response
The message to examine. This must be a response message.

function result
The status code as defined by RFC 2616, section 6.1.1.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageGetTypeID

Returns the Core Foundation type identifier for the CFHTTPMessage opaque type.

```
CFTypeID CFHTTPMessageGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHTTPMessage opaque type.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageIsHeaderComplete

Determines whether a message header is complete.

```
Boolean CFHTTPMessageIsHeaderComplete (
   CFHTTPMessageRef message
);
```

Parameters

message
The message to verify.

function result
TRUE if the message header is complete, otherwise FALSE.

Discussion
After calling `CFHTTPMessageAppendBytes` (page 44), call this function to see if the message header is complete.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageIsRequest

Returns a boolean indicating whether the CFHTTPMessage is a request or a response.

```
extern Boolean CFHTTPMessageIsRequest(CFHTTPMessageRef message);
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageSetBody

Sets the body of a `CFHTTPMessage` object.

```
void CFHTTPMessageSetBody (
   CFHTTPMessageRef message,
   CFDataRef bodyData
);
```

Parameters

message
The message to modify.

bodyData
The data that is to be set as the body of the message.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageSetHeaderFieldValue

Sets the value of a header field in an HTTP message.

```
void CFHTTPMessageSetHeaderFieldValue (
   CFHTTPMessageRef message,
   CFStringRef headerField,
   CFStringRef value
);
```

Parameters

message

The message to modify.

headerField

The header field to set.

value

The value to set.

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPMessage.h

Data Types

CFHTTPMessageRef

An opaque reference representing an HTTP message.

```
typedef struct __CFHTTPMessage *CFHTTPMessageRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
CFHTTPMessage.h

Constants

CFHTTP Version Constants

Sets the HTTP version in a `CFHTTPMessage` request or response object.

```
const CFStringRef kCFHTTPVersion1_0;
const CFStringRef kCFHTTPVersion1_1;
```

Constants

kCFHTTPVersion1_0

Specifies HTTP version 1.0.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPVersion1_1`
Specifies HTTP version 1.1.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

Discussion
The HTTP version constants are used when you call `CFHTTPMessageCreateRequest` (page 51) and `CFHTTPMessageCreateResponse` (page 52) to create a request or response message.

Declared In
`CFNetwork/CFHTTPMessage.h`

Authentication Schemes

Constants used to specify the desired authentication scheme for a request.

```
extern const CFStringRef kCFHTTPAuthenticationSchemeBasic;
extern const CFStringRef kCFHTTPAuthenticationSchemeDigest;
extern const CFStringRef kCFHTTPAuthenticationSchemeNTLM;
extern const CFStringRef kCFHTTPAuthenticationSchemeNegotiate;
extern const CFStringRef kCFHTTPAuthenticationSchemeKerberos;
```

Constants
`kCFHTTPAuthenticationSchemeBasic`
Request the HTTP basic authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeDigest`
Request the HTTP digest authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNTLM`
Request the HTTP NTLM authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNegotiate`
Request the HTTP Negotiate authentication scheme.

Available in iOS 2.0 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeKerberos`
Request the HTTP Kerberos authentication scheme.

Available in iOS 3.2 and later.

Declared in `CFHTTPMessage.h`.

CFHTTPStream Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHTTPStream.h
Companion guides	Getting Started with Networking CFNetwork Programming Guide

Overview

This document describes the CFStream functions for working with HTTP connections.

Functions

CFReadStreamCreateForHTTPRequest

Creates a read stream for a CFHTTP request message.

```
CFReadStreamRef CFReadStreamCreateForHTTPRequest (
   CFAllocatorRef alloc,
   CFHTTPMessageRef request
);
```

Parameters

alloc

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

request

A CFHTTP request message whose body and headers have been set.

Return Value

A new read stream, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

This function creates a read stream and associates it with the specified request. Automatic redirection is disabled by default. After creating the read stream, you can call `CFReadStreamGetError` at any time to check the status of the stream. You may want to call `CFHTTPReadStreamSetRedirectsAutomatically` to enable automatic redirection, or `CFHTTPReadStreamSetProxy` to set the name and port number for a proxy. To serialize the request and send it, call `CFReadStreamOpen`.

If the body of the request is too long to keep in memory, call `CFReadStreamCreateForStreamedHTTPRequest` (page 58) instead of this function.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPStream.h`

CFReadStreamCreateForStreamedHTTPRequest

Creates a read stream for a CFHTTP request message object whose body is too long to keep in memory.

```
CFReadStreamRef CFReadStreamCreateForStreamedHTTPRequest (
   CFAllocatorRef alloc,
   CFHTTPMessageRef requestHeaders,
   CFReadStreamRef requestBody
);
```

Parameters

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

requestHeaders
A CFHTTP request header.

requestBody
Read stream reference for the request body.

Return Value
A new read stream, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function creates a read stream for the response to the *requestHeaders* plus *requestBody*. Call this function instead of `CFReadStreamCreateForHTTPRequest` (page 57) when the body of the request is so long that you do not want it to be resident in memory.

Because streams cannot be reset, read streams created this way cannot be enabled for autoredirection.

If the Content-Length header is set in *requestHeaders*, it is assumed that the length is correct and that *requestBody* will report end-of-stream after precisely Content-Length bytes have been read from it. If the Content-Length header is not set, the chunked transfer-encoding will be added to *requestHeaders*, and bytes read from *requestBody* will be transmitted chunked. The body of *requestHeaders* is ignored.

After creating the read stream, you can call `CFReadStreamGetError` at any time to check the status of the stream. You may want to call `CFHTTPReadStreamSetProxy` to set the name and port number for a proxy. To serialize the request and send it, call `CFReadStreamOpen`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFHTTPStream.h`

Constants

CFStream HTTP Error Constants

Error codes that a read stream for an HTTP request may return.

```
typedef enum {
    kCFStreamErrorHTTPParseFailure = -1,
    kCFStreamErrorHTTPRedirectionLoop = -2,
    kCFStreamErrorHTTPBadURL = -3
} CFStreamErrorHTTP;
```

Constants

`kCFStreamErrorHTTPParseFailure`

A parsing error occurred while an incoming message was being deserialized and appended to a message object. The headers of the incoming message may be formatted improperly.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamErrorHTTPRedirectionLoop`

A redirection loop has been detected.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamErrorHTTPBadURL`

The URL is not properly formatted.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`<CFNetwork/CFHTTPStream.h>`

CFStream HTTP Property Constants

Constants for setting and copying CFStream HTTP properties.

```
const CFStringRef kCFStreamPropertyHTTPAttemptPersistentConnection;
const CFStringRef kCFStreamPropertyHTTPFinalURL;
extern const CFStringRef kCFStreamPropertyHTTPFinalRequest;
const CFStringRef kCFStreamPropertyHTTPProxy;
const CFStringRef kCFStreamPropertyHTTPRequestBytesWrittenCount;
const CFStringRef kCFStreamPropertyHTTPResponseHeader;
const CFStringRef kCFStreamPropertyHTTPShouldAutoredirect;
```

Constants

`kCFStreamPropertyHTTPAttemptPersistentConnection`

HTTP Attempt Persistent Connection property. The value of this property is a CFBoolean. If this property is set to `kCFBooleanTrue`, the HTTP stream looks for an appropriate existing persistent connection to use. If it cannot find one, the HTTP stream will try to create one.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPFinalURL`

HTTP Final URL property. A value of type CFURL containing the final HTTP URL. This value differs from the URL in the original HTTP request if an autoredirection occurred. This property cannot be set.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPFinalRequest`

HTTP Final Request property. A value of type CFHTTPMessage containing the final message transmitted by the stream after all modifications (including authentication, connection policy, redirects, and so on) have been made. This property cannot be set.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPProxy`

HTTP Proxy property. To cause an HTTP CFStream to use an HTTP proxy, set the value of this property to a CFDictionary that includes at least one host/port pair described in "CFStream SOCKS Proxy Key Constants" in *CFStream Reference*. SystemConfiguration returns a CFDictionary that is usable without modification.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPProxyHost`

HTTP Proxy Host property. If an HTTP CFStream is using an HTTP proxy, the value of this property is a CFString containing the host name or IP number of the proxy server.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPProxyPort`

HTTP Proxy Host property. If an HTTP CFStream is using an HTTP proxy, the value of this property is a CFNumber containing the port number of the proxy server.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPRequestBytesWrittenCount`
: HTTP Request Bytes Written property. This property can only be retrieved; it cannot be set. The value of this property is a CFNumber containing the number of body bytes that have been written to the server thus far. HTTP header bytes are not included in the count. You can use this property to track the progress of HTTP uploads that take a substantial amount of time to complete.

  Available in iOS 2.0 and later.

  Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPResponseHeader`
: HTTP Response Header property. When copied by `CFReadStreamCopyProperty`, the header of an HTTP response message is returned.

  Available in iOS 2.0 and later.

  Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPSProxyHost`
: HTTPS Proxy Host property. If a CFStream is using an HTTPS proxy, the value of this property is a CFString containing the host name or IP number of the proxy server.

  Available in iOS 2.0 and later.

  Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPSProxyPort`
: HTTPS Proxy Host property. If a CFStream is using an HTTPS proxy, the value of this property is a CFNumber containing the port number of the proxy server.

  Available in iOS 2.0 and later.

  Declared in `CFHTTPStream.h`.

`kCFStreamPropertyHTTPShouldAutoredirect`
: HTTP Should Auto Redirect property. Set this property to `kCFBooleanTrue` to enable autoredirection; set this property to `kCFBooleanFalse` to disable autoredirection.

  Available in iOS 2.0 and later.

  Declared in `CFHTTPStream.h`.

Discussion
The CFStream HTTP property constants are used to specify the HTTP property to set when calling `CFReadStreamSetProperty` or `CFWriteStreamSetProperty`. They are also used to specify the HTTP property to copy when calling `CFReadStreamCopyProperty` or `CFWriteStreamCopyProperty`.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`<CFNetwork/CFHTTPStream.h>`

Error Domains

Error domains specific to `CFHTTPStream` calls.

```
extern const SInt32 kCFStreamErrorDomainHTTP;
```

Constants

`kCFStreamErrorDomainHTTP`

Error domain that returns errors associated with the `CFHTTPStream` layer.

Available in iOS 2.0 and later.

Declared in `CFHTTPStream.h`.

Discussion

To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFNetDiagnostics Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFNetDiagnostics.h
Companion guide	CFNetwork Programming Guide

Overview

The CFNetDiagnostics opaque type allows you to diagnose network-related problems.

Functions by Task

Creating a net diagnostics object

`CFNetDiagnosticCreateWithStreams` (page 64)
Creates a network diagnostic object from a pair of CFStreams.

`CFNetDiagnosticCreateWithURL` (page 65)
Creates a CFNetDiagnosticRef from a CFURLRef.

CFNetDiagnostics Functions

`CFNetDiagnosticSetName` (page 67)
Overrides the displayed application name.

`CFNetDiagnosticDiagnoseProblemInteractively` (page 66)
Opens a Network Diagnostics window.

`CFNetDiagnosticCopyNetworkStatusPassively` (page 64)
Gets a network status value.

Functions

CFNetDiagnosticCopyNetworkStatusPassively

Gets a network status value.

```
CFNetDiagnosticStatus CFNetDiagnosticCopyNetworkStatusPassively (
   CFNetDiagnosticRef details,
   CFStringRef *description
);
```

Parameters

details

CFNetDiagnosticRef, created by `CFNetDiagnosticCreateWithStreams` (page 64) or `CFNetDiagnosticCreateWithURL` (page 65), for which the Network Diagnostics status is to be obtained.

description

If not `NULL`, upon return contains a localized string containing a description of the current network status. Ownership follows the Create Rule.

Return Value

A network status value.

Discussion

This function returns a status value that can be used to display basic information about the connection, and optionally gets a localized string containing a description of the current network status.

This function is guaranteed not to generate network activity.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticCreateWithStreams

Creates a network diagnostic object from a pair of CFStreams.

```
CFNetDiagnosticRef CFNetDiagnosticCreateWithStreams (
   CFAllocatorRef alloc,
   CFReadStreamRef readStream,
   CFWriteStreamRef writeStream
);
```

Parameters

alloc

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

readStream

Reference to a read stream whose connection has failed, or `NULL` if you do not want the CFNetDiagnosticRef to have a read stream.

writeStream

Reference to a write stream whose connection has failed, or `NULL` if you do not want the CFNetDiagnosticRef to have a write stream.

function result

CFNetDiagnosticRef that you can pass to `CFNetDiagnosticDiagnoseProblemInteractively` (page 66) or `CFNetDiagnosticCopyNetworkStatusPassively` (page 64). Ownership follows the Create Rule.

Discussion

This function uses references to a read steam and a write stream (or just a read stream or just a write stream) to create a reference to an instance of a CFNetDiagnostic object. You can pass the reference to `CFNetDiagnosticDiagnoseProblemInteractively` (page 66) to open a Network Diagnostics window or to `CFNetDiagnosticCopyNetworkStatusPassively` (page 64) to get a description of the connection referenced by `readStream` and `writeStream`.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticCreateWithURL

Creates a CFNetDiagnosticRef from a CFURLRef.

```
CFNetDiagnosticRef CFNetDiagnosticCreateWithURL (
   CFAllocatorRef alloc,
   CFURLRef url
);
```

Parameters

alloc

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

url

CFURLRef that refers to the failed connection.

Return Value
CFNetDiagnosticRef that you can pass to `CFNetDiagnosticDiagnoseProblemInteractively` (page 66) or `CFNetDiagnosticCopyNetworkStatusPassively` (page 64). Ownership follows the Create Rule.

Discussion
This function uses a URL to create a reference to an instance of a CFNetDiagnostic object. You can pass the reference to `CFNetDiagnosticDiagnoseProblemInteractively` (page 66) to open a Network Diagnostics window or to `CFNetDiagnosticCopyNetworkStatusPassively` (page 64) to get a description of the connection referenced by `readStream` and `writeStream`.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetDiagnostics.h`

CFNetDiagnosticDiagnoseProblemInteractively

Opens a Network Diagnostics window.

```
CFNetDiagnosticStatus CFNetDiagnosticDiagnoseProblemInteractively (
   CFNetDiagnosticRef details
);
```

Parameters

details

A network diagnostics object, created by `CFNetDiagnosticCreateWithStreams` (page 64) or `CFNetDiagnosticCreateWithURL` (page 65), for which the window is to be opened.

Return Value
`CFNetDiagnosticNoErr` if no error occurred, or `CFNetDiagnosticErr` if an error occurred that prevented this call from completing successfully.

Discussion
This function opens the Network Diagnostics window and returns immediately once the window is open.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetDiagnostics.h`

CFNetDiagnosticSetName

Overrides the displayed application name.

```
void CFNetDiagnosticSetName (
   CFNetDiagnosticRef details,
   CFStringRef name
);
```

Parameters

details

The network diagnostics object for which the application name is to be set.

name

Name that is to be set.

Discussion

Frameworks requiring that an application name be displayed to the user derive the application name from the bundle identifier of the currently running application, in that application's localization. If you want to override the derived application name, use this function to set the name that is displayed.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetDiagnostics.h`

Data Types

CFNetDiagnosticRef

An opaque reference representing a CFNetDiagnostic.

```
typedef struct __CFNetDiagnostic* CFNetDiagnosticRef;
```

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticStatus

A CFIndex type that is used to return status values from `CFNetDiagnostic` status and diagnostic functions. For a list of possible values, see "CFNetDiagnosticStatusValues Constants" (page 68).

```
typedef CFIndex                         CFNetDiagnosticStatus;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetDiagnostics.h`

Constants

CFNetDiagnosticStatusValues Constants

Constants for diagnostic status values.

```
enum CFNetDiagnosticStatusValues {
    kCFNetDiagnosticNoErr = 0,
    kCFNetDiagnosticErr = -66560L,
    kCFNetDiagnosticConnectionUp = -66559L,
    kCFNetDiagnosticConnectionIndeterminate = -66558L,
    kCFNetDiagnosticConnectionDown = -66557L
};
typedef enum CFNetDiagnosticStatusValues CFNetDiagnosticStatusValues;
```

Constants

`kCFNetDiagnosticNoErr`
No error occurred but there is no status.
Available in iOS 2.0 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticErr`
An error occurred that prevented the call from completing.
Available in iOS 2.0 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionUp`
The connection appears to be working.
Available in iOS 2.0 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionIndeterminate`
The status of the connection is not known.
Available in iOS 2.0 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionDown`
The connection does not appear to be working.
Available in iOS 2.0 and later.
Declared in `CFNetDiagnostics.h`.

Discussion
Diagnostic status values are returned by `CFNetDiagnosticDiagnoseProblemInteractively` (page 66) and `CFNetDiagnosticCopyNetworkStatusPassively` (page 64).

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetwork/CFNetDiagnostics.h`

CFNetServices Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFNetServices.h
Companion guides	Bonjour Overview CFNetwork Programming Guide NSNetServices and CFNetServices Programming Guide

Overview

The CFNetServices API is part of Bonjour, Apple's implementation of zero-configuration networking (ZEROCONF). The CFNetServices API allows you to register a network service, such as a printer or file server, so that it can be found by name or browsed for by service type and domain. Applications can use the CFNetServices API to discover the services that are available on the network and to find all access information — such as name, IP address, and port number — needed to use each service.

In effect, Bonjour registration and discovery combine the functions of a local DNS server and AppleTalk, allowing applications to provide the kind of user-friendly browsing available in the AppleTalk Chooser using open protocols, such as Multicast DNS (mDNS). Bonjour gives applications easy access to services over local IP networks without requiring the service to support an AppleTalk stack, and without requiring a DNS server on the local network.

For a full description of Bonjour, see *Bonjour Overview*.

Functions by Task

Creating net service objects

`CFNetServiceCreate` (page 80)
Creates an instance of a Network Service object.

`CFNetServiceCreateCopy` (page 81)
Creates a copy of a CFNetService object.

`CFNetServiceMonitorCreate` (page 87)
Creates an instance of a NetServiceMonitor object that watches for record changes.

`CFNetServiceBrowserCreate` (page 73)
Creates an instance of a Network Service browser object.

CFNetServices Functions

`CFNetServiceBrowserInvalidate` (page 75)
Invalidates an instance of a Network Service browser object.

`CFNetServiceBrowserScheduleWithRunLoop` (page 75)
Schedules a CFNetServiceBrowser on a run loop.

`CFNetServiceBrowserSearchForDomains` (page 76)
Searches for domains.

`CFNetServiceBrowserSearchForServices` (page 77)
Searches a domain for services of a specified type.

`CFNetServiceBrowserStopSearch` (page 78)
Stops a search for domains or services.

`CFNetServiceBrowserUnscheduleFromRunLoop` (page 79)
Unschedules a CFNetServiceBrowser from a run loop and mode.

`CFNetServiceCancel` (page 79)
Cancels a service registration or a service resolution.

`CFNetServiceCreateDictionaryWithTXTData` (page 82)
Uses TXT record data to create a dictionary.

`CFNetServiceCreateTXTDataWithDictionary` (page 82)
Flattens a set of key/value pairs into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 96).

`CFNetServiceGetAddressing` (page 83)
Gets the IP addressing from a CFNetService.

`CFNetServiceGetTargetHost` (page 85)
Queries a CFNetService for its target hosts.

`CFNetServiceGetDomain` (page 84)
Gets the domain from a CFNetService.

`CFNetServiceGetName` (page 84)
Gets the name from a CFNetService.

`CFNetServiceGetPortNumber` (page 85)
This function gets the port number from a CFNetService.

`CFNetServiceGetTXTData` (page 86)
Queries a network service for the contents of its TXT records.

`CFNetServiceGetType` (page 87)
Gets the type from a CFNetService.

`CFNetServiceMonitorInvalidate` (page 89)
Invalidates an instance of a Network Service monitor object.

`CFNetServiceMonitorScheduleWithRunLoop` (page 89)
Schedules a CFNetServiceMonitor on a run loop.

`CFNetServiceMonitorStart` (page 90)
Starts monitoring.

`CFNetServiceMonitorStop` (page 91)
Stops a CFNetServiceMonitor.

`CFNetServiceMonitorUnscheduleFromRunLoop` (page 92)
Unschedules a CFNetServiceMonitor from a run loop.

`CFNetServiceRegisterWithOptions` (page 92)
Makes a CFNetService available on the network.

`CFNetServiceResolveWithTimeout` (page 93)
Gets the IP address or addresses for a CFNetService.

`CFNetServiceScheduleWithRunLoop` (page 94)
Schedules a CFNetService on a run loop.

`CFNetServiceSetClient` (page 95)
Associates a callback function with a CFNetService or disassociates a callback function from a CFNetService.

`CFNetServiceSetTXTData` (page 96)
Sets the TXT record for a CFNetService.

`CFNetServiceUnscheduleFromRunLoop` (page 96)
Unschedules a CFNetService from a run loop.

Getting the net service type IDs

`CFNetServiceGetTypeID` (page 87)
Gets the Core Foundation type identifier for the Network Service object.

`CFNetServiceMonitorGetTypeID` (page 89)
Gets the Core Foundation type identifier for all CFNetServiceMonitor instances.

`CFNetServiceBrowserGetTypeID` (page 74)
Gets the Core Foundation type identifier for the Network Service browser object.

Functions

CFNetServiceBrowserCreate

Creates an instance of a Network Service browser object.

```
CFNetServiceBrowserRef CFNetServiceBrowserCreate (
   CFAllocatorRef alloc,
   CFNetServiceBrowserClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

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

clientCB
Callback function that is to be called when domains and services are found; cannot be `NULL`. For details, see `CFNetServiceBrowserClientCallBack` (page 97).

clientContext
Context information to be used when clientCB is called; cannot be NULL. For details, see CFNetServiceClientContext (page 100).

Return Value
A new browser object, or NULL if the instance could not be created. Ownership follows the Create Rule.

Discussion
This function creates an instance of a Network Service browser object, called a CFNetServiceBrowser, that can be used to search for domains and for services.

To use the resulting CFNetServiceBrowser in asynchronous mode, call CFNetServiceBrowserScheduleWithRunLoop (page 75). Then call CFNetServiceBrowserSearchForDomains (page 76) and CFNetServiceBrowserSearchForServices (page 77) to use the CFNetServiceBrowser to search for services and domains, respectively. The callback function specified by clientCB is called from a run loop to pass search results to your application. The search continues until you stop the search by calling CFNetServiceBrowserStopSearch (page 78).

If you do not call CFNetServiceBrowserScheduleWithRunLoop (page 75), searches with the resulting CFNetServiceBrowser are made in synchronous mode. Calls made to CFNetServiceBrowserSearchForDomains (page 76) and CFNetServiceBrowserSearchForServices (page 77) block until there are search results, in which case the callback function specified by clientCB is called, until the search is are stopped by calling CFNetServiceBrowserStopSearch (page 78) from another thread, or an error occurs.

To shut down a CFNetServiceBrowser that is running in asynchronous mode, call CFNetServiceBrowserUnscheduleFromRunLoop (page 79), followed by CFNetServiceBrowserInvalidate (page 75), and then CFNetServiceBrowserStopSearch (page 78).

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
CFNetServices.h

CFNetServiceBrowserGetTypeID

Gets the Core Foundation type identifier for the Network Service browser object.

```
CFTypeID CFNetServiceBrowserGetTypeID ();
```

Return Value
The type ID.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
CFNetServices.h

CFNetServiceBrowserInvalidate

Invalidates an instance of a Network Service browser object.

```
void CFNetServiceBrowserInvalidate (
   CFNetServiceBrowserRef browser
);
```

Parameters

browser
The CFNetServiceBrowser to invalidate, obtained by a previous call to CFNetServiceBrowserCreate (page 73).

Discussion
This function invalidates the specified instance of a Network Service browser object. Any searches using the specified instance that are in progress when this function is called are stopped. An invalidated browser cannot be scheduled on a run loop and its callback function is never called.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetServiceBrowserRef at the same time.

Availability
Available in iOS 2.0 and later.

Declared In
CFNetServices.h

CFNetServiceBrowserScheduleWithRunLoop

Schedules a CFNetServiceBrowser on a run loop.

```
void CFNetServiceBrowserScheduleWithRunLoop (
   CFNetServiceBrowserRef browser,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

browser
The CFNetServiceBrowser that is to be scheduled on a run loop; cannot be NULL.

runLoop
The run loop on which the browser is to be scheduled; cannot be NULL.

runLoopMode
The mode on which to schedule the browser; cannot be NULL.

Discussion
This function schedules the specified CFNetServiceBrowser on the run loop, thereby placing the browser in asynchronous mode. The run loop will call the browser's callback function to deliver the results of domain and service searches. The caller is responsible for ensuring that at least one of the run loops on which the browser is scheduled is being run.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserSearchForDomains

Searches for domains.

```
Boolean CFNetServiceBrowserSearchForDomains (
   CFNetServiceBrowserRef browser,
   Boolean registrationDomains,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser, obtained by previously calling `CFNetServiceBrowserCreate` (page 73), that is to perform the search; cannot be `NULL`.

registrationDomains

`TRUE` to search for only registration domains; `FALSE` to search for domains that can be browsed for services. For this version of the CFNetServices API, the registration domain is the local domain maintained by the mDNS responder running on the same machine as the calling application.

error

A pointer to a `CFStreamError` structure, that, if an error occurs, will be set to the error and the error's domain and passed to your callback function. Pass `NULL` if you don't want to receive the error that may occur as a result of this particular call.

Return Value
`TRUE` if the search was started (asynchronous mode); `FALSE` if another search is already in progress for this CFNetServiceBrowser or if an error occurred.

Discussion
This function uses a CFNetServiceBrowser to search for domains. The search continues until the search is canceled by calling `CFNetServiceBrowserStopSearch` (page 78). If `registrationDomains` is `TRUE`, this function searches only for domains in which services can be registered. If `registrationDomains` is `FALSE`, this function searches for domains that can be browsed for services. When a domain is found, the callback function specified when the CFNetServiceBrowser was created is called and passed an instance of a CFStringRef containing the domain that was found.

In asynchronous mode, this function returns `TRUE` if the search was started. Otherwise, it returns `FALSE`.

In synchronous mode, this function blocks until the search is stopped by calling `CFNetServiceBrowserStopSearch` (page 78) from another thread, in which case it returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

For any one CFNetServiceBrowser, only one domain search or one service search can be in progress at the same time.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceBrowserSearchForServices

Searches a domain for services of a specified type.

```
Boolean CFNetServiceBrowserSearchForServices (
   CFNetServiceBrowserRef browser,
   CFStringRef domain,
   CFStringRef serviceType,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser, obtained by previously calling `CFNetServiceBrowserCreate` (page 73), that is to perform the search; cannot be `NULL`.

domain

The domain to search for the service type; cannot be `NULL`. To get the domains that are available for searching, call `CFNetServiceBrowserSearchForDomains` (page 76).

type

The service type to search for; cannot be `NULL`. For a list of valid service types, see http://www.iana.org/assignments/port-numbers.

error

A pointer to a `CFStreamError` structure, that, if an error occurs, will be set to the error and the error's domain and passed to your callback function. Pass `NULL` if you don't want to receive the error that may occur as a result of this particular call.

Return Value

`TRUE` if the search was started (asynchronous mode); `FALSE` if another search is already in progress for this CFNetServiceBrowser or if an error occurred.

Discussion

This function searches the specified domain for services that match the specified service type. The search continues until the search is canceled by calling `CFNetServiceBrowserStopSearch` (page 78). When a match is found, the callback function specified when the CFNetServiceBrowser was created is called and passed an instance of a CFNetService representing the service that was found.

In asynchronous mode, this function returns `TRUE` if the search was started. Otherwise, it returns `FALSE`.

In synchronous mode, this function blocks until the search is stopped by calling `CFNetServiceBrowserStopSearch` (page 78) from another thread, in which case this function returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

For any one CFNetServiceBrowser, only one domain search or one service search can be in progress at the same time.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserStopSearch

Stops a search for domains or services.

```
void CFNetServiceBrowserStopSearch (
   CFNetServiceBrowserRef browser,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser that was used to start the search; cannot be `NULL`.

error

A pointer to a `CFStreamError` structure that will be passed to the callback function associated with this CFNetServiceBrowser (if the search is being conducted in asynchronous mode) or that is pointed to by the `error` parameter when `CFNetServiceBrowserSearchForDomains` (page 76) or `CFNetServiceBrowserSearchForServices` (page 77) returns (if the search is being conducted in synchronous mode). Set the `domain` field to `kCFStreamErrorDomainCustom` and the `error` field to an appropriate value.

Discussion
This functions stops a search started by a previous call to `CFNetServiceBrowserSearchForDomains` (page 76) or `CFNetServiceBrowserSearchForServices` (page 77). For asynchronous and synchronous searches, calling this function causes the callback function associated with the CFNetServiceBrowser to be called once for each domain or service found. If the search is asynchronous, `error` is passed to the callback function. If the search is synchronous, calling this function causes `CFNetServiceBrowserSearchForDomains` or `CFNetServiceBrowserSearchForServices` to return `FALSE`. If the `error` parameter for either call pointed to a `CFStreamError` structure, the `CFStreamError` structure contains the error code and the error code's domain as set when this function was called.

Special Considerations

This function is thread safe.

If you are stopping an asynchronous search, before calling this function, call `CFNetServiceBrowserUnscheduleFromRunLoop` (page 79), followed by `CFNetServiceBrowserInvalidate` (page 75).

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserUnscheduleFromRunLoop

Unschedules a CFNetServiceBrowser from a run loop and mode.

```
void CFNetServiceBrowserUnscheduleFromRunLoop (
   CFNetServiceBrowserRef browser,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

browser
The CFNetServiceBrowser that is to be unscheduled; cannot be `NULL`.

runLoop
The run loop; cannot be `NULL`.

runLoopMode
The mode from which the browser is to be unscheduled; cannot be `NULL`.

Discussion
Call this function to shut down a browser that is running asynchronously. To complete the shutdown, call `CFNetServiceBrowserInvalidate` (page 75) followed by `CFNetServiceBrowserStopSearch` (page 78).

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceCancel

Cancels a service registration or a service resolution.

```
void CFNetServiceCancel (
   CFNetServiceRef theService
);
```

Parameters

theService
The CFNetService, obtained by previously calling `CFNetServiceCreate` (page 80), for which a registration or a resolution is to be canceled.

Discussion
This function cancels service registrations, started by `CFNetServiceRegister`, thereby making the service unavailable. It also cancels service resolutions, started by `CFNetServiceResolve`.

If you are shutting down an asynchronous service, you should first call `CFNetServiceUnscheduleFromRunLoop` (page 96) and `CFNetServiceSetClient` (page 95) with `clientCB` set to `NULL`. Then call this function.

If you are shutting down a synchronous service, call this function from another thread.

This function also cancels service resolutions. You would want to cancel a service resolution if your callback function has received an IP address that you've successfully used to connect to the service. In addition, you might want to cancel a service resolution if the resolution is taking longer than a user would want to wait or if the user canceled the operation.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceCreate

Creates an instance of a Network Service object.

```
CFNetServiceRef CFNetServiceCreate (
   CFAllocatorRef alloc,
   CFStringRef domain,
   CFStringRef serviceType,
   CFStringRef name,
   SInt32 port
);
```

Parameters

alloc

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

domain

The domain in which the CFNetService is to be registered; cannot be `NULL`. Call `CFNetServiceBrowserCreate` (page 73) and `CFNetServiceBrowserSearchForDomains` (page 76) to get the registration domain.

type

The type of service being registered; cannot be `NULL`. For a list of valid service types, see http://www.iana.org/assignments/port-numbers.

name

A unique name if the instance will be used to register a service. The name will become part of the instance name in the DNS records that will be created when the service is registered. If the instance will be used to resolve a service, the name should be the name of the machine or service that will be resolved.

port

Local IP port, in host byte order, on which this service accepts connections. Pass zero to get placeholder service. With a placeholder service, the service will not be discovered by browsing, but a name conflict will occur if another client tries to register the same name. Most applications do not need to use placeholder service.

Return Value
A new net service object, or `NULL` if the instance could not be created. Ownership follows the Create Rule.

Discussion
If the service depends on information in DNS TXT records, call `CFNetServiceSetProtocolSpecificInformation`.

If the CFNetService is to run in asynchronous mode, call `CFNetServiceSetClient` (page 95) to prepare the service for running in asynchronous mode. Then call `CFNetServiceScheduleWithRunLoop` (page 94) to schedule the service on a run loop. Then call `CFNetServiceRegister` to make the service available.

If the CFNetService is to run in synchronous mode, call `CFNetServiceRegister`.

To terminate a service that is running in asynchronous mode, call `CFNetServiceCancel` (page 79) and `CFNetServiceUnscheduleFromRunLoop` (page 96).

To terminate a service that is running in synchronous mode, call `CFNetServiceCancel` (page 79).

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceCreateCopy

Creates a copy of a CFNetService object.

```
CFNetServiceRef CFNetServiceCreateCopy (
   CFAllocatorRef alloc,
   CFNetServiceRef service
);
```

Parameters

alloc

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

service

CFNetServiceRef to be copied; cannot be `NULL`. If `service` is not a valid CFNetServiceRef, the behavior of this function is undefined.

Return Value
Copy of `service`, including all previously resolved data, or `NULL` if `service` could not be copied. Ownership follows the Create Rule.

Discussion
This function creates a copy of the CFNetService specified by `service`.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceCreateDictionaryWithTXTData

Uses TXT record data to create a dictionary.

```
CFDictionaryRef CFNetServiceCreateDictionaryWithTXTData (
   CFAllocatorRef alloc,
   CFDataRef txtRecord
);
```

Parameters

`alloc`

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

`txtRecord`

TXT record data as returned by `CFNetServiceGetTXTData` (page 86).

Return Value
A dictionary containing the key/value pairs parsed from `txtRecord`, or `NULL` if `txtRecord` cannot be parsed. Each key in the dictionary is a CFString object, and each value is a CFData object. Ownership follows the Create Rule.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceCreateTXTDataWithDictionary

Flattens a set of key/value pairs into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 96).

```
CFDataRef CFNetServiceCreateTXTDataWithDictionary (
   CFAllocatorRef alloc,
   CFDictionaryRef keyValuePairs
);
```

Parameters

alloc

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

keyValuePairs

CFDictionaryRef containing the key/value pairs that are to be placed in a TXT record. Each key must be a CFStringRef and each value should be a CFDataRef or a CFStringRef. (See the discussion below for additional information about values that are CFStringRefs.) This function fails if any other data types are provided. The length of a key and its value should not exceed 255 bytes.

Return Value

A CFData object containing the flattened form of *keyValuePairs*, or `NULL` if the dictionary could not be flattened. Ownership follows the Create Rule.

Discussion

This function flattens the key/value pairs in the dictionary specified by `keyValuePairs` into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 96). Note that this function is not a general purpose function for flattening CFDictionaryRefs.

The keys in the dictionary referenced by `keyValuePairs` must be CFStringRefs and the values must be CFDataRefs. Any values that are CFStringRefs are converted to CFDataRefs representing the flattened UTF-8 bytes of the string. The types of the values are not encoded in the CFDataRefs, so any CFStringRefs that are converted to CFDataRefs remain CFDataRefs when the CFDataRef produced by this function is processed by `CFNetServiceCreateDictionaryWithTXTData` (page 82).

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetAddressing

Gets the IP addressing from a CFNetService.

```
CFArrayRef CFNetServiceGetAddressing (
   CFNetServiceRef theService
);
```

Parameters

theService

The CFNetService whose IP addressing is to be obtained; cannot be `NULL`.

Return Value
A CFArray containing a CFDataRef for each IP address returned, or `NULL`. Each CFDataRef consists of a `sockaddr` structure containing the IP address of the service. This function returns `NULL` if the service's addressing is unknown because `CFNetServiceResolve` has not been called for `theService`.

Discussion
This function gets the IP addressing from a CFNetService. Typically, the CFNetService was obtained by calling `CFNetServiceBrowserSearchForServices` (page 77). Before calling this function, call `CFNetServiceResolve` to update the CFNetService with its IP addressing.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceGetDomain

Gets the domain from a CFNetService.

```
CFStringRef CFNetServiceGetDomain (
   CFNetServiceRef theService
);
```

Parameters

`theService`

The CFNetService whose domain is to be obtained; cannot be `NULL`.

Return Value
A CFString object containing the domain of the CFNetService.

Discussion
This function gets the domain from a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceGetName

Gets the name from a CFNetService.

```
CFStringRef CFNetServiceGetName (
   CFNetServiceRef theService
);
```

Parameters

theService

The CFNetService whose name is to be obtained; cannot be `NULL`.

Return Value

A CFString object containing the name of the service represented by the CFNetService.

Discussion

This function gets the name from a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetPortNumber

This function gets the port number from a CFNetService.

```
extern SInt32 CFNetServiceGetPortNumber(
    CFNetServiceRef theService);
```

Parameters

theService

The CFNetService whose protocol-specific information is to be obtained; cannot be `NULL`. Note that in order to get protocol-specific information, you must resolve `theService` by calling `CFNetServiceResolve` or `CFNetServiceResolveWithTimeout` (page 93) before calling this function.

Return Value

A CFString object containing the protocol-specific information, or `NULL` if there is no information.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetTargetHost

Queries a CFNetService for its target hosts.

```
CFStringRef CFNetServiceGetTargetHost (
   CFNetServiceRef theService
);
```

Parameters

theService

Network service to be queried.

Return Value

The target host name of the machine providing the service or `NULL` is of the service's target host is not known. (The target host will not be known if it has not been resolved.)

Special Considerations

This function is thread safe, but the target host name is not safe if the service is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetTXTData

Queries a network service for the contents of its TXT records.

```
CFDataRef CFNetServiceGetTXTData (
   CFNetServiceRef theService
);
```

Parameters

theService

Reference for the network service whose TXT record data is to be obtained; cannot be `NULL`. Note that in order to get TXT record data, you must resolve `theService` by calling `CFNetServiceResolve` or `CFNetServiceResolveWithTimeout` (page 93) before calling this function.

Return Value

CFDataRef object containing the requested TXT data and suitable for passing to `CFNetServiceCreateDictionaryWithTXTData` (page 82), or `NULL` if the service's TXT data has not been resolved.

Discussion

This function gets the data from the service's TXT records.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetType

Gets the type from a CFNetService.

```
CFStringRef CFNetServiceGetType (
   CFNetServiceRef theService
);
```

Parameters

theService
The CFNetService whose type is to be obtained; cannot be `NULL`.

Return Value

A CFString object containing the type from a CFNetService.

Discussion

This function gets the type of a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetTypeID

Gets the Core Foundation type identifier for the Network Service object.

```
CFTypeID CFNetServiceGetTypeID ();
```

Return Value

The type ID.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorCreate

Creates an instance of a NetServiceMonitor object that watches for record changes.

```
CFNetServiceMonitorRef CFNetServiceMonitorCreate (
   CFAllocatorRef alloc,
   CFNetServiceRef theService,
   CFNetServiceMonitorClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

alloc

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

theService

CFNetService to be monitored.

clientCB

Pointer to callback function that is to be called when a record associated with `theService` changes; cannot be `NULL`.

clientContext

Pointer to user-defined contextual information that is to be passed to the callback specified by `clientCB` when the callback is called; cannot be `NULL`. For details, see CFNetServiceClientContext (page 100).

Return Value

A new instance of a CFNetServiceMonitor, or `NULL` if the monitor could not be created. Ownership follows the Create Rule.

Discussion

This function creates a CFNetServiceMonitor that watches for changes in records associated with `theService`.

If the CFNetServiceMonitor is to run in asynchronous mode, call `CFNetServiceMonitorScheduleWithRunLoop` (page 89) to schedule the monitor on a run loop. Then call `CFNetServiceMonitorStart` (page 90) to start monitoring. When a change occurs, the callback function specified by `clientCB` will be called. For details, see CFNetServiceMonitorClientCallBack (page 99).

If the CFNetServiceMonitor is to run in synchronous mode, call `CFNetServiceMonitorStart` (page 90).

To stop a monitor that is running in asynchronous mode, call `CFNetServiceMonitorStop` (page 91) and `CFNetServiceMonitorUnscheduleFromRunLoop` (page 92).

To stop a monitor that is running in synchronous mode, call `CFNetServiceMonitorStop` (page 91).

If you no longer need to monitor record changes, call `CFNetServiceMonitorStop` (page 91) to stop the monitor and then call `CFNetServiceMonitorInvalidate` (page 89)to invalidate the monitor so it cannot be used again. Then call `CFRelease` to release the memory associated with CFNetServiceMonitorRef.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorGetTypeID

Gets the Core Foundation type identifier for all CFNetServiceMonitor instances.

```
CFTypeID CFNetServiceMonitorGetTypeID ();
```

Return Value
The type ID.

Special Considerations

This function is thread safe.

Version Notes
Introduced in Mac OS X v10.4.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorInvalidate

Invalidates an instance of a Network Service monitor object.

```
void CFNetServiceMonitorInvalidate (
   CFNetServiceMonitorRef monitor
);
```

Parameters

monitor
CFNetServiceMonitor to invalidate; cannot be `NULL`.

Discussion
This function invalidates the specified Network Service monitor so that it cannot be used again. Before you call this function, you should call `CFNetServiceMonitorStop` (page 91). If the monitor has not already been stopped, this function stops the monitor for you.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorScheduleWithRunLoop

Schedules a CFNetServiceMonitor on a run loop.

```
void CFNetServiceMonitorScheduleWithRunLoop (
   CFNetServiceMonitorRef monitor,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService

The CFNetServiceMonitor that is to be scheduled on a run loop; cannot be `NULL`.

runLoop

The run loop on which the monitor is to be scheduled; cannot be `NULL`.

runLoopMode

The mode on which to schedule the monitor; cannot be `NULL`.

Discussion

Schedules the specified monitor on a run loop, which places the monitor in asynchronous mode. The caller is responsible for ensuring that at least one of the run loops on which the monitor is scheduled is being run.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorStart

Starts monitoring.

```
Boolean CFNetServiceMonitorStart (
   CFNetServiceMonitorRef monitor,
   CFNetServiceMonitorType recordType,
   CFStreamError *error
);
```

Parameters

monitor

CFNetServiceMonitor, created by calling `CFNetServiceMonitorCreate` (page 87), that is to be started.

recordType

CFNetServiceMonitorType that specified the type of record to monitor. For possible values, see CFNetServiceMonitorType Constants (page 103).

error

Pointer to a `CFStreamError` structure. If an error occurs, on output, the structure's `domain` field will be set to the error code's domain and the `error` field will be set to an appropriate error code. Set this parameter to `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous monitor was started successfully. `FALSE` if an error occurred when starting an asynchronous or synchronous monitor, or if `CFNetServiceMonitorStop` (page 91) was called for an synchronous monitor.

Discussion
This function starts monitoring for changes to records of the type specified by `recordType`. If a monitor is already running for the service associated with the specified CFNetServiceMonitorRef, this function returns `FALSE`.

For synchronous monitors, this function blocks until the monitor is stopped by calling `CFNetServiceMonitorStop` (page 91), in which case, this function returns `FALSE`.

For asynchronous monitors, this function returns `TRUE` or `FALSE`, depending on whether monitoring starts successfully.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorStop

Stops a CFNetServiceMonitor.

```
void CFNetServiceMonitorStop (
   CFNetServiceMonitorRef monitor,
   CFStreamError *error
);
```

Parameters

monitor

CFNetServiceMonitor, started by calling `CFNetServiceMonitorStart` (page 90), that is to be stopped.

error

Pointer to a `CFStreamError` structure or `NULL`. For synchronous monitors, set the `error` field of this structure to the non-zero value you want to be set in the `CFStreamError` structure when `CFNetServiceMonitorStart` (page 90) returns. Note that when it returns, `CFNetServiceMonitorStart` returns `FALSE`. If the monitor was started asynchronously, set the `error` field to the non-zero value you want the monitor's callback to receive when it is called. If this parameter is `NULL`, default values for the `CFStreamError` structure are used: the domain is set to `kCFStreamErrorDomainNetServices` and the error code is set to `kCFNetServicesErrorCancel`.

Discussion
This function stops the specified monitor. Call `CFNetServiceMonitorStart` (page 90) if you want to start monitoring again.

If you want to stop monitoring and no longer need to monitor record changes, call `CFNetServiceMonitorInvalidate` (page 89) instead of this function.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
CFNetServices.h

CFNetServiceMonitorUnscheduleFromRunLoop

Unschedules a CFNetServiceMonitor from a run loop.

```
void CFNetServiceMonitorUnscheduleFromRunLoop (
   CFNetServiceMonitorRef monitor,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

monitor
The CFNetServiceMonitor that is to be unscheduled; cannot be NULL.

runLoop
The run loop; cannot be NULL.

runLoopMode
The mode from which the monitor is to be unscheduled; cannot be NULL.

Discussion
Unschedules the specified monitor from the specified run loop and mode. Call this function to shut down a monitor that is running asynchronously.

To change a monitor so that it cannot be scheduled and so that its callback will never be called, call CFNetServiceMonitorInvalidate (page 89).

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
CFNetServices.h

CFNetServiceRegisterWithOptions

Makes a CFNetService available on the network.

```
Boolean CFNetServiceRegisterWithOptions (
   CFNetServiceRef theService,
   CFOptionFlags options,
   CFStreamError *error
);
```

Parameters

theService
Network service to register; cannot be NULL. The registration will fail if the service doesn't have a domain, a type, a name, and an IP address.

options

Bit flags for specifying registration options. Currently, the only registration option is `kCFNetServiceFlagNoAutoRename`. For details, see CFNetService Registration Options (page 102).

error

Pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous service registration was started; `FALSE` if an asynchronous or synchronous registration failed or if a synchronous registration was canceled.

Discussion

If the service is to run in asynchronous mode, you must call `CFNetServiceSetClient` (page 95) to associate a callback function with this CFNetService before calling this function.

When registering a service that runs in asynchronous mode, this function returns `TRUE` if the service contains all of the required attributes and the registration process can start. If the registration process completes successfully, the service is available on the network until you shut down the service by calling `CFNetServiceUnscheduleFromRunLoop` (page 96), `CFNetServiceSetClient` (page 95), and `CFNetServiceCancel` (page 79). If the service does not contain all of the required attributes or if the registration process does not complete successfully, this function returns `FALSE`.

When registering a service that runs in synchronous mode, this function blocks until an error occurs, in which case this function returns `FALSE`. Until this function returns `FALSE`, the service is available on the network. To force this function to return `FALSE`, thereby shutting down the service, call `CFNetServiceCancel` (page 79) from another thread.

The `options` parameter is a bit flag for specifying service registration options. Currently, `kCFNetServiceFlagNoAutoRename` is the only supported registration option. If this bit is set and a service of the same name is running, the registration will fail. If this bit is not set and a service of the same name is running, the service that is being registered will be renamed automatically by appending `(n)` to the service name, where *n* is a number that is incremented until the service can be registered with a unique name.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceResolveWithTimeout

Gets the IP address or addresses for a CFNetService.

```
Boolean CFNetServiceResolveWithTimeout (
   CFNetServiceRef theService,
   CFTimeInterval timeout,
   CFStreamError *error
);
```

Parameters

theService

The CFNetService to resolve; cannot be `NULL`. The resolution will fail if the service doesn't have a domain, a type, and a name.

timeout

Value of type `CFTimeInterval` specifying the maximum amount of time allowed to perform the resolution. If the resolution is not performed within the specified amount of time, a timeout error will be returned. If `timeout` is less than or equal to zero, an infinite amount of time is allowed.

error

Pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous service resolution was started or if a synchronous service resolution updated the CFNetService; `FALSE` if an asynchronous or synchronous resolution failed or timed out, or if a synchronous resolution was canceled.

Discussion

This function updates the specified CFNetService with the IP address or addresses associated with the service. Call `CFNetServiceGetAddressing` (page 83) to get the addresses.

When resolving a service that runs in asynchronous mode, this function returns `TRUE` if the CFNetService has a domain, type, and name, and the underlying resolution process was started. Otherwise, this function returns `FALSE`. Once started, the resolution continues until it is canceled by calling `CFNetServiceCancel` (page 79).

When resolving a service that runs in synchronous mode, this function blocks until the CFNetService is updated with at least one IP address, until an error occurs, or until `CFNetServiceCancel` (page 79) is called.

Special Considerations

This function is thread safe.

If the service will be used in asynchronous mode, you must call `CFNetServiceSetClient` (page 95) before calling this function.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceScheduleWithRunLoop

Schedules a CFNetService on a run loop.

```
void CFNetServiceScheduleWithRunLoop (
   CFNetServiceRef theService,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService

The CFNetService that is to be scheduled on a run loop; cannot be `NULL`.

runLoop

The run loop on which the service is to be scheduled; cannot be `NULL`.

runLoopMode

The mode on which to schedule the service; cannot be `NULL`.

Discussion

Schedules the specified service on a run loop, which places the service in asynchronous mode. The caller is responsible for ensuring that at least one of the run loops on which the service is scheduled is being run.

Special Considerations

This function is thread safe.

Before calling this function, call `CFNetServiceSetClient` (page 95) to prepare a CFNetService for use in asynchronous mode.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceSetClient

Associates a callback function with a CFNetService or disassociates a callback function from a CFNetService.

```
Boolean CFNetServiceSetClient (
   CFNetServiceRef theService,
   CFNetServiceClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

theService

The CFNetService; cannot be `NULL`.

clientCB

The callback function that is to be associated with this CFNetService. If you are shutting down the service, set `clientCB` to `NULL` to disassociate from this CFNetService the callback function that was previously associated.

clientContext

Context information to be used when `clientCB` is called; cannot be `NULL`.

Return Value

`TRUE` if the client was set; otherwise, `FALSE`.

Discussion
The callback function specified by `clientCB` will be called to report IP addresses (in the case of `CFNetServiceResolve`) or to report registration errors (in the case of `CFNetServiceRegister`).

Special Considerations

This function is thread safe.

For a CFNetService that will operate asynchronously, call this function and then call `CFNetServiceScheduleWithRunLoop` (page 94) to schedule the service on a run loop. Then call `CFNetServiceRegister` or `CFNetServiceResolve`.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceSetTXTData

Sets the TXT record for a CFNetService.

```
Boolean CFNetServiceSetTXTData (
   CFNetServiceRef theService,
   CFDataRef txtRecord
);
```

Parameters

`theService`
CFNetServiceRef for which a TXT record is to be set; cannot be `NULL`.

`txtRecord`
Contents of the TXT record that is to be set. The contents must not exceed 1450 bytes.

Return Value
`TRUE` if the TXT record was set; otherwise, `FALSE`.

Discussion
This function sets a TXT record for the specified service. If the service is currently registered on the network, the record is broadcast. Setting a TXT record on a service that is still being resolved is not allowed.

Special Considerations

This function is thread safe.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceUnscheduleFromRunLoop

Unschedules a CFNetService from a run loop.

```
void CFNetServiceUnscheduleFromRunLoop (
   CFNetServiceRef theService,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService

The CFNetService that is to be unscheduled; cannot be `NULL`.

runLoop

The run loop; cannot be `NULL`.

runLoopMode

The mode from which the service is to be unscheduled; cannot be `NULL`.

Discussion

Unschedules the specified service from the specified run loop and mode. Call this function to shut down a service that is running asynchronously. To complete the shutdown, call `CFNetServiceSetClient` (page 95) and set `clientCB` to `NULL`. Then call `CFNetServiceCancel` (page 79).

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

Callbacks

CFNetServiceBrowserClientCallBack

Defines a pointer to the callback function for a CFNetServiceBrowser.

```
typedef void (*CFNetServiceBrowserClientCallBack) (
   CFNetServiceBrowserRef browser,
   CFOptionFlags flags,
   CFTypeRef domainOrService,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceBrowserClientCallBack`, you would declare it like this:

```
void MyNetServiceBrowserClientCallBack (
   CFNetServiceBrowserRef browser,
   CFOptionFlags flags,
   CFTypeRef domainOrService,
   CFStreamError* error,
   void* info);
```

Parameters

browser

The CFNetServiceBrowser associated with this callback function.

flags

Flags conveying additional information. The `kCFNetServiceFlagIsDomain` bit is set if `domainOrService` contains a domain; if this bit is not set, `domainOrService` contains a CFNetService instance. For additional bit values, see CFNetServiceBrowserClientCallBack Bit Flags (page 102).

domainOrService

A string containing a domain name if this callback function is being called as a result of calling `CFNetServiceBrowserSearchForDomains` (page 76), or a CFNetService instance if this callback function is being called as a result calling `CFNetServiceBrowserSearchForServices` (page 77).

error

A pointer to a `CFStreamError` structure whose `error` field may contain an error code.

info

User-defined context information. The value of `info` is the same as the value of the `info` field of the `CFNetServiceClientContext` (page 100) structure that was provided when `CFNetServiceBrowserCreate` (page 73) was called to create the CFNetServiceBrowser associated with this callback function.

Discussion

The callback function for a CFNetServiceBrowser is called one or more times when domains or services are found as the result of calling `CFNetServiceBrowserSearchForDomains` (page 76) and `CFNetServiceBrowserSearchForServices` (page 77).

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceClientCallBack

Defines a pointer to the callback function for a CFNetService.

```
typedef void (*CFNetServiceClientCallBack) (
   CFNetServiceRef theService,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceClientCallBack`, you would declare it like this:

```
void MyNetServiceClientCallBack (
   CFNetServiceRef theService,
   CFStreamError* error,
   void* info);
```

Parameters

theService

CFNetService associated with this callback function.

error

Pointer to a `CFStreamError` structure whose `error` field contain may contain an error code.

info

User-defined context information. The value of `info` is the same as the value of the `info` field of the `CFNetServiceClientContext` (page 100) structure that was provided when `CFNetServiceSetClient` (page 95) was called for the CFNetService associated with this callback function.

Discussion

Your callback function will be called when there are results of resolving a CFNetService to report or when there are registration errors to report. In the case of resolution, if the service has more than one IP address, your callback will be called once for each address.

Availability

Available in iOS 2.0 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorClientCallBack

Defines a pointer to the callback function that is to be called when a monitored record type changes.

```
typedef void (*CFNetServiceMonitorClientCallBack) (
   CFNetServiceMonitorRef theMonitor,
   CFNetServiceRef theService,
   CFNetServiceMonitorType typeInfo,
   CFDataRef rdata,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceMonitorClientCallBack`, you would declare it like this:

```
void MyNetServiceMonitorClientCallBack (
   CFNetServiceMonitorRef theMonitor,
   CFNetServiceRef theService,
   CFNetserviceMonitorType typeInfo,
   CFDataRef rdata,
   CFStreamError *error,
   void *info);
```

Parameters

theMonitor

CFNetServiceMonitor for which the callback is being called.

theService

CFNetService for which the callback is being called.

typeInfo

Type of record that changed. For possible values, see CFNetServiceMonitorType Constants (page 103).

rdata

Contents of the record that changed.

error

Pointer to `CFStreamError` structure whose `error` field contains an error code if an error occurred.

info

Arbitrary pointer to the user-defined data that was specified in the `info` field of the `CFNetServiceClientContext` structure when the monitor was created by `CFNetServiceMonitorCreate` (page 87).

Discussion
The callback function will be called when the monitored record type changes or when the monitor is stopped by calling `CFNetServiceMonitorStop` (page 91).

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

Data Types

CFNetServiceBrowserRef

An opaque reference representing a CFNetServiceBrowser.

```
typedef struct __CFNetServiceBrowser* CFNetServiceBrowserRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceClientContext

A structure provided when a CFNetService is associated with a callback function or when a CFNetServiceBrowser is created.

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

Fields
`version`

Version number for this structure. Currently the only valid value is zero.

`info`

Arbitrary pointer to user-allocated memory containing user-defined data that is associated with the service, browser, or monitor and is passed to their respective callback functions. The data must be valid for as long as the CFNetService, CFNetServiceBrowser, or CFNetServiceMonitor is valid. Set this field to `NULL` if your callback function doesn't want to receive user-defined data.

`retain`
: The callback used to add a retain for the service or browser using `info` for the life of the service or browser. This callback may be used for temporary references the service or browser needs to take. This callback returns the actual `info` pointer so it can be stored in the service or browser. This field can be `NULL`.

`release`
: Callback that removes a retain previously added for the service or browser on the `info` pointer. This field can be `NULL`, but setting this field to `NULL` may result in memory leaks.

`copyDescription`
: Callback used to create a descriptive string representation of the data pointed to by `info`. In implementing this function, return a reference to a CFString object that describes your allocator and some characteristics of your user-defined data, which is used by `CFCopyDescription()`. You can set this field to `NULL`, in which case Core Foundation will provide a rudimentary description.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorRef

An opaque reference for a service monitor.

```
typedef struct __CFNetServiceMonitor* CFNetServiceMonitorRef;
```

Discussion
Service monitor references are used to monitor record changes on a CFNetServiceRef.

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

CFNetServiceRef

An opaque reference representing a CFNetService.

```
typedef struct __CFNetService* CFNetServiceRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CFNetServices.h`

Constants

CFNetService Registration Options

Bit flags used when registering a service.

```
enum {
    kCFNetServiceFlagNoAutoRename = 1
};
```

Constants

`kCFNetServiceFlagNoAutoRename`

Causes registrations to fail if a name conflict occurs.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetwork/CFNetServices.h`

CFNetServiceBrowserClientCallBack Bit Flags

Bit flags providing additional information about the result returned when a client's `CFNetServiceBrowserClientCallBack` function is called.

```
enum {
kCFNetServiceFlagMoreComing = 1,
kCFNetServiceFlagIsDomain = 2,
kCFNetServiceFlagIsDefault = 4,
kCFNetServiceFlagIsRegistrationDomain = 4, /* For compatibility  */
kCFNetServiceFlagRemove = 8
};
```

Constants

`kCFNetServiceFlagMoreComing`

If set, a hint that the client's callback function will be called again soon; therefore, the client should not do anything time-consuming, such as updating the screen.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagIsDomain`

If set, the results pertain to a search for domains. If not set, the results pertain to a search for services.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagIsDefault`
If set, the resulting domain is the default registration or browse domain, depending on the context. For this version of the CFNetServices API, the default registration domain is the local domain. In previous versions of this API, this constant was `kCFNetServiceFlagIsRegistrationDomain`, which is retained for backward compatibility.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagRemove`
If set, the client should remove the result item instead of adding it.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

Discussion
See `CFNetServiceBrowserClientCallBack` for additional information.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetwork/CFNetServices.h`

CFNetServiceMonitorType Constants

Record type specifier used to tell a service monitor the type of record changes to watch for.

```
enum {
kCFNetServiceMonitorTXT = 1
} typedef enum CFNetServiceMonitorType CFNetServiceMonitorType;
```

Constants
`kCFNetServiceMonitorTXT`
Watch for TXT record changes.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetwork/CFNetServices.h`

CFNetService Error Constants

Error codes that may be returned by CFNetServices functions or passed to CFNetServices callback functions.

```
typedef enum {
    kCFNetServicesErrorUnknown = -72000,
    kCFNetServicesErrorCollision = -72001,
    kCFNetServicesErrorNotFound = -72002,
    kCFNetServicesErrorInProgress = -72003,
    kCFNetServicesErrorBadArgument = -72004,
    kCFNetServicesErrorCancel = -72005,
    kCFNetServicesErrorInvalid = -72006,
    kCFNetServicesErrorTimeout = -72007
} CFNetServicesError;
```

Constants

`kCFNetServicesErrorUnknown`
An unknown CFNetService error occurred.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorCollision`
An attempt was made to use a name that is already in use.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorNotFound`
Not used.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorInProgress`
A search is already in progress.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorBadArgument`
A required argument was not provided.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorCancel`
The search or service was canceled.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorInvalid`
Invalid data was passed to a CFNetServices function.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

`kCFNetServicesErrorTimeout`
Resolution failed because the timeout was reached.
Available in iOS 2.0 and later.
Declared in `CFNetServices.h`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetwork/CFNetServices.h`

Error Domains

Error domains.

```
extern const SInt32 kCFStreamErrorDomainMach;
extern const SInt32 kCFStreamErrorDomainNetServices;
```

Constants

`kCFStreamErrorDomainMach`

Error domain returning errors reported by Mach. For more information, see the header file `/usr/include/mach/error.h`.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

`kCFStreamErrorDomainNetServices`

Error domain returning errors reported by the service discovery APIs. These errors are only returned if you use the `CFNetServiceBrowser` API or any APIs introduced in Mac OS X v10.4 or later.

Available in iOS 2.0 and later.

Declared in `CFNetServices.h`.

CFStream Socket Additions

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFSocketStream.h
Companion guide	CFNetwork Programming Guide

Overview

This document describes the `CFStream` functions for working with sockets. It is part of the CFSocketStream API.

Functions by Task

Creating Socket Pairs

`CFStreamCreatePairWithSocketToCFHost` (page 109)
Creates readable and writable streams connected to a given `CFHost` object.

`CFStreamCreatePairWithSocketToNetService` (page 110)
Creates a pair of streams for a CFNetService.

Obtaining Errors

`CFSocketStreamSOCKSGetError` (page 108)
This function gets error codes in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

`CFSocketStreamSOCKSGetErrorSubdomain` (page 108)
Gets the error subdomain associated with errors in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

Functions

CFSocketStreamSOCKSGetError

This function gets error codes in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

```
SInt32 CFSocketStreamSOCKSGetError(CFStreamError* error);
```

Parameters

error

The error value to decode.

Discussion

Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetError` (page 108) returns the error code itself, which must be interpreted in the context of the result of a call to `CFSocketStreamSOCKSGetErrorSubdomain` (page 108). Possible return values (beyond subdomain-specific values such as client versions and HTTP error codes) are listed in "CFStream Errors" (page 119).

Availability

Available in iOS 2.0 and later.

Declared In

`CFSocketStream.h`

CFSocketStreamSOCKSGetErrorSubdomain

Gets the error subdomain associated with errors in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

```
SInt32 CFSocketStreamSOCKSGetErrorSubdomain(CFStreamError* error);
```

Parameters

error

The error value to decode.

Discussion

Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 108) returns an identifier that tells which layer of the protocol stack produced the error. The possible values are listed in "Error Subdomains" (page 118). With this information, you can interpret the error codes returned by `CFSocketStreamSOCKSGetError` (page 108).

Availability
Available in iOS 2.0 and later.

Declared In
`CFSocketStream.h`

CFStreamCreatePairWithSocketToCFHost

Creates readable and writable streams connected to a given `CFHost` object.

```
void CFStreamCreatePairWithSocketToCFHost (
   CFAllocatorRef alloc,
   CFHostRef host,
   SInt32 port,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

alloc
The allocator to use to allocate memory for the `CFReadStream` and `CFWriteStream` objects. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

host
A `CFHost` object to which the streams are connected. If unresolved, the host will be resolved prior to connecting.

port
The TCP port number to which the socket streams should connect.

readStream
Upon return, contains a `CFReadStream` object connected to the host *host* on port `port`, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a readable stream. Ownership follows the Create Rule.

writeStream
Upon return, contains a `CFWriteStream` object connected to the host *host* on port `port`, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a writable stream. Ownership follows the Create Rule.

Discussion
The streams do not open a connection to the specified host until one of the streams is opened.

Most properties are shared by both streams. Setting the property for one stream automatically sets the property for the other.

Availability
Available in iOS 2.0 and later.

Declared In
`CFSocketStream.h`

CFStreamCreatePairWithSocketToNetService

Creates a pair of streams for a CFNetService.

```
void CFStreamCreatePairWithSocketToNetService (
   CFAllocatorRef alloc,
   CFNetServiceRef service,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

alloc

The allocator to use to allocate memory for the `CFReadStream` and `CFWriteStream` objects. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

service

Reference to the `CFNetService` to which the streams are to be connected. If the service is not resolved, the service will be resolved before the streams are connected.

readstream

Upon return, contains a `CFReadStream` object connected to the service specified by *service*, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a readable stream. Ownership follows the Create Rule.

writeStream

Upon return, contains a `CFWriteStream` object connected to the service specified by *service*, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a writable stream. Ownership follows the Create Rule.

Discussion

Read and write operations on sockets can block. To prevent blocking, you can call `CFReadStreamSetClient` and `CFWriteStreamSetClient` to register to receive stream-related event notifications. Then call `CFReadStreamScheduleWithRunLoop` and `CFWriteStreamScheduleWithRunLoop` to schedule the stream on a run loop for receiving stream-related event notifications. Then call `CFReadStreamOpen` and `CFWriteStreamOpen` to open each stream.

Special Considerations

This function is thread safe.

Availability

Available in iOS 2.0 and later.

Declared In

`CFSocketStream.h`

Constants

CFStream Property Keys

Constants for `CFStream` property keys

```
const CFStringRef kCFStreamPropertyShouldCloseNativeSocket;
const CFStringRef kCFStreamPropertySocketSecurityLevel;
const CFStringRef kCFStreamPropertySOCKSProxy;
const CFStringRef kCFStreamPropertySSLPeerCertificates;
const CFStringRef kCFStreamPropertySSLPeerTrust;
const CFStringRef kCFStreamPropertySSLSettings;
const CFStringRef kCFStreamPropertyProxyLocalByPass;
const CFStringRef kCFStreamPropertySocketRemoteHost;
const CFStringRef kCFStreamPropertySocketRemoteNetService;
const CFStringRef kCFStreamNetworkServiceType;
```

Constants

`kCFStreamPropertyShouldCloseNativeSocket`

Should Close Native Socket property key.

If set to `kCFBooleanTrue`, the stream will close and release the underlying native socket when the stream is released. If set to `kCFBooleanFalse`, the stream will not close and release the underlying native socket when the stream is released. If a stream is created with a native socket, the default value of this property is `kCFBooleanFalse`. This property is only available for socket streams. It can be set by calling `CFReadStreamSetProperty` and `CFWriteStreamSetProperty`, and it can be copied by `CFReadStreamCopyProperty` and `CFWriteStreamCopyProperty`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketNativeHandle`

Socket Native Handle property key.

Causes `CFReadStreamCopyProperty` or `CFWriteStreamCopyProperty` to return `CFData` object that contains the native handle for a socket stream. This property is only available for socket streams.

Available in iOS 2.0 and later.

Declared in `CFStream.h`.

`kCFStreamPropertySocketSecurityLevel`

Socket Security Level property key.

See CFStream Socket Security Level Constants (page 115) for specific security level constants to use.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySSLPeerCertificates`

SSL Peer Certificates property key for copy operations, which return a `CFArray` object containing `SecCertificateRef` objects.

For more information, see `SSLGetPeerCertificates` in `Security/SecureTransport.h`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySSLPeerTrust`

SSL Peer Trust property key for copy operations, which return a `SecTrustRef` object containing the result of the SSL handshake.

For more information, see `SSLCopyPeerTrust` in `Security/SecureTransport.h`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSProxy`
SOCKS proxy property key.

To set a `CFStream` object to use a SOCKS proxy, call `CFReadStreamSetProperty` or `CFWriteStreamSetProperty` with the property name set to `kCFStreamPropertySOCKSProxy` and its value set to a `CFDictionary` object having at minimum a `kCFStreamPropertySOCKSProxyHost` key and a `kCFStreamPropertySOCKSProxyPort` key. For information on these keys, see CFStream SOCKS Proxy Key Constants (page 116). SystemConfiguration returns a CFDictionary for SOCKS proxies that is usable without modification.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySSLSettings`
SSL Settings property key for set operations.

The key's value is a `CFDictionary` object containing security settings. For information on the dictionary's keys and values, see CFStream Property SSL Settings Constants (page 112). By default, there are no security settings.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertyProxyLocalBypass`
Proxy Local Bypass property key.

The key's value is `CFBoolean` object whose value indicates whether local hostnames should be subject to proxy handling.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketRemoteHost`
The key's value is a `CFHostRef` for the remote host if it is known. If not, its value is `NULL`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketRemoteNetService`
The key's value is a `CFNetServiceRef` for the remote network service if it is known. If not, its value is `NULL`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamNetworkServiceType`
The type of service for the stream. Providing the service type allows the system to properly handle certain attributes of the stream, including routing and suspension behavior. Most streams do not need to set this property. See "`Stream Service Types`" (page 117) for a list of possible values.

Available in iOS 4.0 and later.

Declared in `CFSocketStream.h`.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream Property SSL Settings Constants

Constants for use in a `CFDictionary` object that is the value of the `kCFStreamPropertySSLSettings` stream property key.

```
const CFStringRef kCFStreamSSLLevel;
const CFStringRef kCFStreamSSLAllowsExpiredCertificates;
const CFStringRef kCFStreamSSLAllowsExpiredRoots;
const CFStringRef kCFStreamSSLAllowsAnyRoot;
const CFStringRef kCFStreamSSLValidatesCertificateChain;
const CFStringRef kCFStreamSSLPeerName;
const CFStringRef kCFStreamSSLCertificates;
const CFStringRef kCFStreamSSLIsServer;
```

Constants

`kCFStreamSSLLevel`

Security property key whose value specifies the stream's security level.

By default, a stream's security level is `kCFStreamSocketSecurityLevelNegotiatedSSL`. For other possible values, see CFStream Socket Security Level Constants (page 115).

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsExpiredCertificates`

Security property key whose value indicates whether expired certificates are allowed.

By default, the value of this key is `kCFBooleanFalse` (expired certificates are not allowed).

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsExpiredRoots`

Security property whose value indicates whether expired root certificates are allowed.

By default, the value of this key is `kCFBooleanFalse` (expired root certificates are not allowed).

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsAnyRoot`

Security property key whose value indicates whether root certificates should be allowed.

By default, the value of this key is `kCFBooleanFalse` (root certificates are not allowed).

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLValidatesCertificateChain`

Security property key whose value indicates whether the certificate chain should be validated.

By default, the value of this key is `kCFBooleanTrue` (the certificate chain should be validated).

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLPeerName`

Security property key whose value overrides the name used for certificate verification.

By default, the host name that was used when the stream was created is used; if no host name was used, no peer name will be used. Set the value of this key to `kCFNull` to prevent name verification.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLCertificates`
: Security property key whose value is a CFArray of SecCertificateRefs except for the first element in the array, which is a SecIdentityRef.

  For more information, see `SSLSetCertificate()` in `Security/SecureTransport.h`.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLIsServer`
: Security property key whose value indicates whether the connection is to act as a server in the SSL process.

  By default, the value of this key is `kCFBooleanFalse` (the connection is not to act as a server). If the value of this key is `kCFBooleanTrue`, the `kCFStreamSSLCertificates` key must contain a valid value.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

Discussion
This enumeration defines the constants for keys in a `CFDictionary` object that is the value of the `kCFStreamPropertySSLSettings` key.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream Socket Security Protocol Constants

Specifies constants for setting the security protocol for a socket stream.

```
typedef enum {
    kCFStreamSocketSecurityNone = 0,
    kCFStreamSocketSecuritySSLv2,
    kCFStreamSocketSecuritySSLv3,
    kCFStreamSocketSecuritySSLv23,
    kCFStreamSocketSecurityTLSv1
} CFStreamSocketSecurityProtocol;
```

Constants
`kCFStreamSocketSecurityNone`
: Specifies that no security protocol be set for a socket stream. (Deprecated. Use `kCFStreamSocketSecurityLevelNone`.)

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv2`
: Specifies that SSL version 2 be set as the security protocol for a socket stream. (Deprecated. Use `kCFStreamSocketSecurityLevelSSLv2`.)

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv3`

Specifies that SSL version 3 be set as the security protocol for a socket stream. (Deprecated. Use `kCFStreamSocketSecurityLevelSSLv3`.)

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv23`

Specifies that SSL version 3 be set as the security protocol for a socket stream pair. If that version is not available, specifies that SSL version 2 be set as the security protocol for a socket stream. (Deprecated. Use `kCFStreamSocketSecurityLevelNegotiatedSSL`.)

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityTLSv1`

Specifies that TLS version 1 be set as the security protocol for a socket stream. (Deprecated. Use `kCFStreamSocketSecurityLevelTLSv1`.)

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

Discussion

This enumeration defines constants for setting the security protocol for a socket stream pair when calling `CFSocketStreamPairSetSecurityProtocol`.

Special Considerations

This enumeration is deprecated in favor of the constants described in CFStream Socket Security Level Constants (page 115).

Declared In

`CFNetwork/CFSocketStream.h`

CFStream Socket Security Level Constants

Constants for setting the security level of a socket stream.

```
const CFStringRef kCFStreamSocketSecurityLevelNone;
const CFStringRef kCFStreamSocketSecurityLevelSSLv2;
const CFStringRef kCFStreamSocketSecurityLevelSSLv3;
const CFStringRef kCFStreamSocketSecurityLevelTLSv1;
const CFStringRef kCFStreamSocketSecurityLevelNegotiatedSSL;
```

Constants

`kCFStreamSocketSecurityLevelNone`

Specifies that no security level be set.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelSSLv2`

Specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelSSLv3`
Specifies that SSL version 3 be set as the security protocol for a socket stream pair.

If SSL version 3 is not available, specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelTLSv1`
Specifies that TLS version 1 be set as the security protocol for a socket stream.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelNegotiatedSSL`
Specifies that the highest level security protocol that can be negotiated be set as the security protocol for a socket stream.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

Discussion
This enumeration defines the preferred constants for setting the security protocol for a socket stream pair when calling `CFReadStreamSetProperty` or `CFWriteStreamSetProperty`.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream SOCKS Proxy Key Constants

Constants for SOCKS Proxy `CFDictionary` keys.

```
const CFStringRef kCFStreamPropertySOCKSProxyHost;
const CFStringRef kCFStreamPropertySOCKSProxyPort;
const CFStringRef kCFStreamPropertySOCKSVersion;
const CFStringRef kCFStreamSocketSOCKSVersion4;
const CFStringRef kCFStreamSocketSOCKSVersion5;
const CFStringRef kCFStreamPropertySOCKSUser;
const CFStringRef kCFStreamPropertySOCKSPassword;
```

Constants
`kCFStreamPropertySOCKSProxyHost`
Constant for the SOCKS proxy host key.

This key contains a CFString object that represents the SOCKS proxy host. Defined to match `kSCPropNetProxiesSOCKSProxy`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSProxyPort`
Constant for the SOCKS proxy host port key.

This key contains a `CFNumberRef` object of type `kCFNumberSInt32Type` whose value represents the port on which the proxy listens.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSVersion`
: Constant for the SOCKS version key.

  Its value must be `kCFStreamSocketSOCKSVersion4` or `kCFStreamSocketSOCKSVersion5` to set SOCKS4 or SOCKS5, respectively. If this key is not present, SOCKS5 is used by default.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSocketSOCKSVersion4`
: Constant used in the `kCFStreamSockerSOCKSVersion` key to specify SOCKS4 as the SOCKS version for the stream.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSocketSOCKSVersion5`
: Constant used in the `kCFStreamSOCKSVersion` key to specify SOCKS5 as the SOCKS version for the stream.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSUser`
: Constant for the key required to set a user name.

  The value is a `CFString` object containing the user's name.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSPassword`
: Constant for the key required to set a user's password.

  The value is a `CFString` object containing the user's password.

  Available in iOS 2.0 and later.

  Declared in `CFSocketStream.h`.

Discussion
When setting the stream's SOCKS Proxy property, the property's value is a `CFDictionary` object containing at minimum the `kCFStreamPropertySOCKSProxyHost` and `kCFStreamPropertySOCKSProxyPort` keys. The dictionary may also contain the other keys described in this section.

Stream Service Types

String constants that specify the service type of a stream.

```
CFStringRef const kCFStreamNetworkServiceTypeVoIP;
```

Constants
`kCFStreamNetworkServiceTypeVoIP`
: Specifies that the stream is providing VoIP service.

  Available in iOS 4.0 and later.

  Declared in `CFSocketStream.h`.

Error Domains

Error domains specific to `CFSocketStream` calls.

```
extern const int kCFStreamErrorDomainSOCKS;
extern const int kCFStreamErrorDomainSSL;
extern const CFIndex kCFStreamErrorDomainWinSock;
```

Constants

`kCFStreamErrorDomainSOCKS`

This domain returns error codes from the SOCKS layer. The errors are described in

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorDomainSSL`

This domain returns error codes associated with the SSL layer. For a list of error codes, see the header `SecureTransport.h` in `Security.framework`.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorDomainWinSock`

When running CFNetwork code on Windows, this domain returns error codes associated with the underlying TCP/IP stack. You should also note that non-networking errors such as `ENOMEM` are delivered through the POSIX domain. See the header `winsock2.h` for relevant error codes.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

Discussion

To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

Error Subdomains

Subdomains used to determine how to interpret an error in the `kCFStreamErrorDomainSOCKS` domain.

```
enum {
    kCFStreamErrorSOCKSSubDomainNone = 0,
    kCFStreamErrorSOCKSSubDomainVersionCode = 1,
    kCFStreamErrorSOCKS4SubDomainResponse = 2,
    kCFStreamErrorSOCKS5SubDomainUserPass = 3,
    kCFStreamErrorSOCKS5SubDomainMethod = 4,
    kCFStreamErrorSOCKS5SubDomainResponse = 5
};
```

Constants

`kCFStreamErrorSOCKSSubDomainNone`

The error code returned is a SOCKS error number.

Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKSSubDomainVersionCode`

The error returned contains the version of SOCKS that the server wishes to use.

Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS4SubDomainResponse`
: The error returned is the status code that the server returned after the last operation.

  Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainUserPass`
: This subdomain returns error codes associated with the last authentication attempt.

  Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainMethod`
: This subdomain returns the server's desired negotiation method.

  Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainResponse`
: This subdomain returns the response code sent by the server when replying to a connection request.

  Available in Mac OS X version 10.5 and later.

Discussion
Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 108) returns an identifier that tells which layer of the protocol stack produced the error. This list of constants contains the possible values that this function will return.

Calling `CFSocketStreamSOCKSGetError` (page 108) returns the actual error code that the subdomain describes.

CFStream Errors

Error codes returned by the `kCFStreamErrorDomainSOCKS` error domain.

```
/* kCFStreamErrorSOCKSSubDomainNone*/
enum {
  kCFStreamErrorSOCKS5BadResponseAddr = 1,
  kCFStreamErrorSOCKS5BadState  = 2,
  kCFStreamErrorSOCKSUnknownClientVersion = 3
};


/* kCFStreamErrorSOCKS4SubDomainResponse*/
enum {
  kCFStreamErrorSOCKS4RequestFailed = 91,
  kCFStreamErrorSOCKS4IdentdFailed = 92,
  kCFStreamErrorSOCKS4IdConflict = 93
};
```

```
/* kCFStreamErrorSOCKS5SubDomainMethod*/
enum {
  kSOCKS5NoAcceptableMethod     = 0xFF
};
```

Constants

`kCFStreamErrorSOCKS5BadResponseAddr`
The address returned is not of a known type. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS5BadState`
The stream is not in a state that allows the requested operation. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain..

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKSUnknownClientVersion`
The SOCKS server rejected access because it does not support connections with the requested SOCKS version. SOCKS client version. You can query the `kCFSOCKSVersionKey` key to find out what version the server requested. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4RequestFailed`
Request rejected by the server or request failed. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4IdentdFailed`
Request rejected by the server because it could not connect to the `identd` daemon on the client. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4IdConflict`
Request rejected by the server because the client program and the `identd` daemon reported different user IDs. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

`kSOCKS5NoAcceptableMethod`
The client and server could not find a mutually agreeable authentication method. This error code is only valid for errors in the `kCFStreamErrorSOCKS5SubDomainMethod` subdomain.

Available in iOS 2.0 and later.

Declared in `CFSocketStream.h`.

Discussion
Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 108) returns an identifier that tells which layer of the protocol stack produced the error.

Calling `CFSocketStreamSOCKSGetError` (page 108) returns the actual error code that the subdomain describes. This list of constants contains the possible values that this function will return. They must be interpreted within the context of the relevant error subdomain.

Other References

CFNetwork Error Codes Reference

Framework:	CoreServices
Declared in	CFNetwork/CFNetworkErrors.h
Companion guide	CFNetwork Programming Guide

Overview

Many functions in the CFNetwork API return error codes to indicate the cause of a failure. This document explains these error codes.

Constants

CFNetworkErrors Constants

This enumeration contains error codes returned under the error domain `kCFErrorDomainCFNetwork`.

```
enum CFNetworkErrors {
  kCFHostErrorHostNotFound      = 1,
  kCFHostErrorUnknown           = 2,

/* SOCKS errors */
  kCFSOCKSErrorUnknownClientVersion = 100,
  kCFSOCKSErrorUnsupportedServerVersion = 101,

/* SOCKS4-specific errors*/
  kCFSOCKS4ErrorRequestFailed   = 110,
  kCFSOCKS4ErrorIdentdFailed    = 111,
  kCFSOCKS4ErrorIdConflict      = 112,
  kCFSOCKS4ErrorUnknownStatusCode = 113,

/* SOCKS5-specific errors*/
  kCFSOCKS5ErrorBadState        = 120,
  kCFSOCKS5ErrorBadResponseAddr = 121,
  kCFSOCKS5ErrorBadCredentials  = 122,
  kCFSOCKS5ErrorUnsupportedNegotiationMethod = 123,
  kCFSOCKS5ErrorNoAcceptableMethod = 124,
```

```
/* FTP errors */
  kCFFTPErrorUnexpectedStatusCode = 200,

/* HTTP errors*/
  kCFErrorHTTPAuthenticationTypeUnsupported = 300,
  kCFErrorHTTPBadCredentials    = 301,
  kCFErrorHTTPConnectionLost    = 302,
  kCFErrorHTTPParseFailure      = 303,
  kCFErrorHTTPRedirectionLoopDetected = 304,
  kCFErrorHTTPBadURL            = 305,
  kCFErrorHTTPProxyConnectionFailure  = 306,
  kCFErrorHTTPBadProxyCredentials     = 307,
  kCFErrorPACFileError          = 308,
  kCFErrorPACFileAuth           = 309,
  kCFErrorHTTPSProxyConnectionFailure = 310,

/* CFURL and CFURLConnection Errors */
  kCFURLErrorUnknown   = -998,
  kCFURLErrorCancelled = -999,
  kCFURLErrorBadURL    = -1000,
  kCFURLErrorTimedOut  = -1001,
  kCFURLErrorUnsupportedURL = -1002,
  kCFURLErrorCannotFindHost = -1003,
  kCFURLErrorCannotConnectToHost    = -1004,
  kCFURLErrorNetworkConnectionLost  = -1005,
  kCFURLErrorDNSLookupFailed        = -1006,
  kCFURLErrorHTTPTooManyRedirects   = -1007,
  kCFURLErrorResourceUnavailable    = -1008,
  kCFURLErrorNotConnectedToInternet = -1009,
  kCFURLErrorRedirectToNonExistentLocation = -1010,
  kCFURLErrorBadServerResponse             = -1011,
  kCFURLErrorUserCancelledAuthentication   = -1012,
  kCFURLErrorUserAuthenticationRequired    = -1013,
  kCFURLErrorZeroByteResource        = -1014,
  kCFURLErrorCannotDecodeRawData     = -1015,
  kCFURLErrorCannotDecodeContentData = -1016,
  kCFURLErrorCannotParseResponse     = -1017,
  kCFURLErrorInternationalRoamingOff = -1018,
  kCFURLErrorCallIsActive               = -1019,
  kCFURLErrorDataNotAllowed             = -1020,
  kCFURLErrorRequestBodyStreamExhausted = -1021,
  kCFURLErrorFileDoesNotExist           = -1100,
  kCFURLErrorFileIsDirectory            = -1101,
  kCFURLErrorNoPermissionsToReadFile    = -1102,
  kCFURLErrorDataLengthExceedsMaximum   = -1103,

/* SSL errors */
  kCFURLErrorSecureConnectionFailed          = -1200,
  kCFURLErrorServerCertificateHasBadDate     = -1201,
  kCFURLErrorServerCertificateUntrusted      = -1202,
  kCFURLErrorServerCertificateHasUnknownRoot = -1203,
  kCFURLErrorServerCertificateNotYetValid    = -1204,
  kCFURLErrorClientCertificateRejected       = -1205,
  kCFURLErrorClientCertificateRequired       = -1206,

  kCFURLErrorCannotLoadFromNetwork           = -2000,
```

```
/* Download and file I/O errors */
  kCFURLErrorCannotCreateFile = -3000,
  kCFURLErrorCannotOpenFile = -3001,
  kCFURLErrorCannotCloseFile = -3002,
  kCFURLErrorCannotWriteToFile = -3003,
  kCFURLErrorCannotRemoveFile = -3004,
  kCFURLErrorCannotMoveFile = -3005,
  kCFURLErrorDownloadDecodingFailedMidStream = -3006,
  kCFURLErrorDownloadDecodingFailedToComplete = -3007,

/* Cookie errors */
  kCFHTTPCookieCannotParseCookieFile = -4000,

/* Errors originating from CFNetServices*/
  kCFNetServiceErrorUnknown     = -72000L,
  kCFNetServiceErrorCollision   = -72001L,
  kCFNetServiceErrorNotFound    = -72002L,
  kCFNetServiceErrorInProgress  = -72003L,
  kCFNetServiceErrorBadArgument = -72004L,
  kCFNetServiceErrorCancel      = -72005L,
  kCFNetServiceErrorInvalid     = -72006L,
  kCFNetServiceErrorTimeout     = -72007L,
  kCFNetServiceErrorDNSServiceFailure = -73000L,
};
typedef enum CFNetworkErrors CFNetworkErrors;
```

Constants

`kCFHostErrorHostNotFound`

Indicates that the DNS lookup failed.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFHostErrorUnknown`

An unknown error occurred (a name server failure, for example). For additional information, you can query the `kCFGetAddrInfoFailureKey` key to obtain the value returned by `getaddrinfo(3)` and look up the value in `/usr/include/netdb.h`.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKSErrorUnknownClientVersion`

The SOCKS server rejected access because it does not support connections with the requested SOCKS version. SOCKS client version. You can query the `kCFSOCKSVersionKey` key to find out what version the server requested.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKSErrorUnsupportedServerVersion`

The version of SOCKS requested by the server is not supported. You can query the `kCFSOCKSVersionKey` key to find out what version the server requested.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS4ErrorRequestFailed`
Request rejected by the server or request failed. This error is specific to SOCKS4.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS4ErrorIdentdFailed`
Request rejected by the server because it could not connect to the `identd` daemon on the client. This error is specific to SOCKS4.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS4ErrorIdConflict`
Request rejected by the server because the client program and the `identd` daemon reported different user IDs. This error is specific to SOCKS4.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS4ErrorUnknownStatusCode`
The status code returned by the server is unknown. This error is specific to SOCKS4.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS5ErrorBadState`
The stream is not in a state that allows the requested operation.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS5ErrorBadResponseAddr`
The address type returned is not supported

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS5ErrorBadCredentials`
The SOCKS server refused the client connection because of bad login credentials.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS5ErrorUnsupportedNegotiationMethod`
The requested method is not supported. You can query the `kCFSOCKSNegotiationMethodKey` key to find out which method the server requested.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKS5ErrorNoAcceptableMethod`
The client and server could not find a mutually agreeable authentication method.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFFTPErrorUnexpectedStatusCode`
The server returned an unexpected status code.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPAuthenticationTypeUnsupported`
: The client and server could not agree on a supported authentication type.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPBadCredentials`
: The credentials provided for an authenticated connection were rejected by the server.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPConnectionLost`
: The connection to the server was dropped. This usually indicates a highly overloaded server.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPParseFailure`
: The HTTP server response could not be parsed.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPRedirectionLoopDetected`
: Too many HTTP redirects occurred before reaching a page that did not redirect the client to another page. This usually indicates a redirect loop.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPBadURL`
: The requested URL could not be retrieved.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPProxyConnectionFailure`
: A connection could not be established to the HTTP proxy.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPBadProxyCredentials`
: The authentication credentials provided for logging into the proxy were rejected.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorPACFileError`
: An error occurred with the proxy autoconfiguration file.

  Available in iOS 2.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorPACFileAuth`
: The authentication credentials provided by the proxy autoconfiguration file were rejected.

  Available in iOS 3.0 and later.

  Declared in `CFNetworkErrors.h`.

`kCFErrorHTTPSProxyConnectionFailure`
A connection could not be established to the HTTPS proxy.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorUnknown`
An unknown error occurred.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCancelled`
The connection was cancelled.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorBadURL`
The connection failed due to a malformed URL.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorTimedOut`
The connection timed out.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorUnsupportedURL`
The connection failed due to an unsupported URL scheme.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotFindHost`
The connection failed because the host could not be found.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotConnectToHost`
The connection failed because a connection cannot be made to the host.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorNetworkConnectionLost`
The connection failed because the network connection was lost.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorDNSLookupFailed`
The connection failed because the DNS lookup failed.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorHTTPTooManyRedirects`
The HTTP connection failed due to too many redirects.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorResourceUnavailable`
The connection's resource is unavailable.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorNotConnectedToInternet`
The connection failed because the device is not connected to the internet.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorRedirectToNonExistentLocation`
The connection was redirected to a nonexistent location.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorBadServerResponse`
The connection received an invalid server response.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorUserCancelledAuthentication`
The connection failed because the user cancelled required authentication.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorUserAuthenticationRequired`
The connection failed because authentication is required.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorZeroByteResource`
The resource retrieved by the connection is zero bytes.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotDecodeRawData`
The connection cannot decode data encoded with a known content encoding.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotDecodeContentData`
The connection cannot decode data encoded with an unknown content encoding.

Available in iOS 3.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotParseResponse`
The connection cannot parse the server's response.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorInternationalRoamingOff`
The connection failed because international roaming is disabled on the device.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCallIsActive`
The connection failed because a call is active.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorDataNotAllowed`
The connection failed because data use is currently not allowed on the device.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorRequestBodyStreamExhausted`
The connection failed because its request's body stream was exhausted.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorFileDoesNotExist`
The file operation failed because the file does not exist.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorFileIsDirectory`
The file operation failed because the file is a directory.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorNoPermissionsToReadFile`
The file operation failed because it does not have permission to read the file.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorDataLengthExceedsMaximum`
The file operation failed because the file is too large.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorSecureConnectionFailed`
The secure connection failed for an unknown reason.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorServerCertificateHasBadDate`
The secure connection failed because the server's certificate has an invalid date.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorServerCertificateUntrusted`
The secure connection failed because the server's certificate is not trusted.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorServerCertificateHasUnknownRoot`
The secure connection failed because the server's certificate has an unknown root.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorServerCertificateNotYetValid`
The secure connection failed because the server's certificate is not yet valid.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorClientCertificateRejected`
The secure connection failed because the client's certificate was rejected.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorClientCertificateRequired`
The secure connection failed because the server requires a client certificate.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotLoadFromNetwork`
The connection failed because it is being required to return a cached resource, but one is not available.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotCreateFile`
The file cannot be created.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotOpenFile`
The file cannot be opened.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotCloseFile`
The file cannot be closed.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotWriteToFile`
The file cannot be written.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotRemoveFile`
The file cannot be removed.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorCannotMoveFile`
The file cannot be moved.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorDownloadDecodingFailedMidStream`
The download failed because decoding of the downloaded data failed mid-stream.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFURLErrorDownloadDecodingFailedToComplete`
The download failed because decoding of the downloaded data failed to complete.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFHTTPCookieCannotParseCookieFile`
The cookie file cannot be parsed.
Available in iOS 3.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorUnknown`
An unknown error occurred.
Available in iOS 2.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorCollision`
An attempt was made to use a name that is already in use.
Available in iOS 2.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorNotFound`
Not used.
Available in iOS 2.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorInProgress`
A new search could not be started because a search is already in progress.
Available in iOS 2.0 and later.
Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorBadArgument`
A required argument was not provided or was not valid.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorCancel`
The search or service was cancelled.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorInvalid`
Invalid data was passed to a CFNetServices function.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorTimeout`
A search failed because it timed out.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFNetServiceErrorDNSServiceFailure`
DNS service discovery returned an error. You can query the `kCFDNSServiceFailureKey` key to find out the error returned by DNS service discovery and look up the code in `/usr/include/dns_ds.h` or *DNS Service Discovery C Reference*.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

Discussion
To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read. These errors are all part of the `kCFErrorDomainCFNetwork` domain.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CFNetwork/CFHost.h`

Property Keys

Keys for calls to property get/set functions such as `CFReadStreamSetProperty` and `CFReadStreamCopyProperty`.

```
extern const CFStringRef kCFGetAddrInfoFailureKey;
extern const CFStringRef kCFSOCKSStatusCodeKey;
extern const CFStringRef kCFSOCKSVersionKey;
extern const CFStringRef kCFSOCKSNegotiationMethodKey;
extern const CFStringRef kCFDNSServiceFailureKey;
extern const CFStringRef kCFFTPStatusCodeKey;
```

Constants

`kCFGetAddrInfoFailureKey`

Querying this key returns the last error code returned by `getaddrinfo(3)` in response to a DNS lookup. To interpret the results, look up the error code in `/usr/include/netdb.h`.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKSStatusCodeKey`

Querying this key returns the last status code sent by the SOCKS server in response to the previous operation.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKSVersionKey`

Querying this key returns the SOCKS version in use by the current connection.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFSOCKSNegotiationMethodKey`

Querying this key returns the negotiation method requested by the SOCKS server.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFDNSServiceFailureKey`

Querying this key returns the last error returned by the DNS resolver libraries in response to the previous operation. To interpret the results, look up the error codes in `/usr/include/dns_sd.h` or *DNS Service Discovery C Reference*.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFFTPStatusCodeKey`

Querying this key returns the last status code sent by the FTP server in response to the previous operation.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

Error Domains

High-level error domains.

```
extern const CFStringRef kCFErrorDomainCFNetwork;
extern const CFStringRef kCFErrorDomainWinSock;
```

Constants

`kCFErrorDomainCFNetwork`

Error domain that returns error codes specific to the CFNetwork stack.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

`kCFErrorDomainWinSock`

Error domain that returns error codes specific to the underlying network layer when running CFNetwork code on Windows.

Available in iOS 2.0 and later.

Declared in `CFNetworkErrors.h`.

Discussion

To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFProxySupport Reference

Framework:	CoreServices
Declared in	CFNetwork/CFProxySupport.h
Companion guide	CFNetwork Programming Guide

Overview

The CFProxySupport API enables you to take advantage of global proxy configuration settings in your application.

The CFProxySupport functions return arrays of dictionaries, where each dictionary describes a single proxy. The arrays represent the order in which the proxies should be tried. In general, you should try to download a URL using the first proxy in the array, try the second proxy if the first one fails, and so on.

Every proxy dictionary has an entry for `kCFProxyTypeKey`. If the type is anything except `kCFProxyTypeAutoConfigurationURL`, the dictionary also has entries for the proxy's host and port (under `kCFProxyHostNameKey` and `kCFProxyPortNumberKey` respectively). If the type is `kCFProxyTypeAutoConfigurationURL`, it has an entry for `kCFProxyAutoConfigurationURLKey`.

The keys for username and password are optional and are present only if the username or password could be extracted from the information passed in (either from the URL itself or from the proxy dictionary supplied). These APIs do not consult any external credential stores such as the Keychain.

Functions

CFNetworkCopyProxiesForAutoConfigurationScript

Executes a proxy autoconfiguration script to determine the best proxy to use to retrieve a specified URL.

```
extern CFArrayRef
CFNetworkCopyProxiesForAutoConfigurationScript(
  CFStringRef   proxyAutoConfigurationScript,
  CFURLRef      targetURL);
```

Parameters

proxyAutoConfigurationScript

A `CFString` containing the code of the autoconfiguration script to execute.

targetURL

The URL your application intends to access.

Availability
Available in iOS 2.0 and later.

Declared In
`CFProxySupport.h`

CFNetworkCopyProxiesForURL

Returns the list of proxies that should be used to download a given URL.

```
extern CFArrayRef
CFNetworkCopyProxiesForURL(
  CFURLRef          url,
  CFDictionaryRef   proxySettings
);
```

Parameters

url

The URL your application intends to access.

proxySettings

A dictionary describing the available proxy settings. The dictionary should be in the format returned by `SystemConfiguration.framework`. (See *System Configuration Framework Reference* for more information.)

Return Value
Returns an array of dictionaries. Each dictionary describes a single proxy. The array is ordered optimally for requesting the URL specified.

Availability
Available in iOS 2.0 and later.

Declared In
`CFProxySupport.h`

CFNetworkCopySystemProxySettings

Returns a CFDictionary containing the current systemwide internet proxy settings.

```
CFDictionaryRef CFNetworkCopySystemProxySettings( void );
```

Discussion
The dictionary returned contains key-value pairs that represent the current internet proxy settings. The keys in this dictionary are defined in "Global Proxy Settings Constants" (page 145).

The caller is responsible for releasing the returned dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
`CFProxySupport.h`

CFNetworkExecuteProxyAutoConfigurationScript

Downloads a proxy autoconfiguration script and executes it.

```
CFRunLoopSourceRef
CFNetworkExecuteProxyAutoConfigurationScript(
  CFStringRef proxyAutoConfigurationScript,
  CFURLRef targetURL,
  CFProxyAutoConfigurationResultCallback cb,
  CFStreamClientContext *clientContext);
```

Parameters

proxyAutoConfigurationScript
: A `CFString` containing the code of the autoconfiguration script to be executed.

targetURL
: The URL that your application intends to eventually download using the proxies.

cb
: A callback to be called when execution of the script is finished.

clientContext
: A stream context containing a client info object and optionally retain and release callbacks for that object.

Discussion

This function returns a run loop source that the caller should schedule. Once execution of the script has completed, the specified callback function is called.

> **Note:** If you want to terminate the request before completion, you should invalidate the run loop source.

Availability

Available in iOS 2.0 and later.

Declared In

`CFProxySupport.h`

CFNetworkExecuteProxyAutoConfigurationURL

Downloads a proxy autoconfiguration script and executes it.

```
extern CFRunLoopSourceRef
CFNetworkExecuteProxyAutoConfigurationURL(
  CFURLRef proxyAutoConfigURL,
  CFURLRef targetURL,
  CFProxyAutoConfigurationResultCallback cb,
  CFStreamClientContext *clientContext);
```

Parameters

proxyAutoConfigURL
: The URL of the autoconfiguration script.

targetURL
: The URL that your application intends to eventually download using the proxies.

cb
: A callback to be called when execution of the script is finished.

clientContext
A stream context containing a client info object and optionally retain and release callbacks for that object.

Discussion
This function returns a run loop source that the caller should schedule. Once downloading and execution of the script has completed, the specified callback function is called.

Note: If you want to terminate the request before completion, you should invalidate the run loop source.

Availability
Available in iOS 2.0 and later.

Declared In
`CFProxySupport.h`

Callbacks

CFProxyAutoConfigurationResultCallback

Callback function called when a proxy autoconfiguration computation has completed.

```
typedef CALLBACK_API_C(
    void,
    CFProxyAutoConfigurationResultCallback
)(
    void *client,
    CFArrayRef proxyList,
    CFErrorRef error
);

void myFunction(
    void *client,
    CFArrayRef proxyList,
    CFErrorRef error
);
```

Parameters

client
The client reference originally passed in the `clientContext` parameter of the `CFNetworkExecuteProxyAutoConfigurationScript` or `CFNetworkExecuteProxyAutoConfigurationURL` call that triggered this callback.

proxyList
The list of proxies returned by the autoconfiguration script. This list is in a format suitable for passing to `CFProxyCopyProxiesForURL` (with the added guarantee that no entries will ever be autoconfiguration URL entries). If an error occurs, this value will be `NULL`.

Note: If you want to keep this list, you must retain it when your callback receives it.

error
An error object that indicates any error that may have occurred. If no error occurred, this value will be NULL.

Availability
Available in iOS 2.0 and later.

Declared In
`CFProxySupport.h`

Constants

Property Keys

Keys for calls to property get/set functions such as `CFReadStreamSetProperty` and `CFReadStreamCopyProperty`.

```
const CFStringRef kCFProxyAutoConfigurationHTTPResponseKey;
const CFStringRef kCFProxyAutoConfigurationJavaScriptKey;
const CFStringRef kCFProxyAutoConfigurationURLKey;
const CFStringRef kCFProxyHostNameKey;
const CFStringRef kCFProxyPasswordKey;
const CFStringRef kCFProxyPortNumberKey;
const CFStringRef kCFProxyTypeKey;
const CFStringRef kCFProxyUsernameKey;
```

Constants
`kCFProxyAutoConfigurationHTTPResponseKey`
A `CFHTTPMessageRef` value found in the user info dictionary of an authentication error returned to the `CFNetworkCopyProxiesForAutoConfigurationScript` (page 139) function or to a `CFProxyAutoConfigurationResultCallback` (page 142) callback.

Available in iOS 3.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyAutoConfigurationJavaScriptKey`
A `CFString` value containing the full JavaScript source for the proxy autoconfiguration (PAC) file. This key is only present for proxies of type `kCFProxyAutoConfigurationJavaScript`.

Available in iOS 4.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyAutoConfigurationURLKey`
A `CFURL` value specifying the location of the proxy autoconfiguration (PAC) file. This key is only present for proxies of type `kCFProxyTypeAutoConfigurationURL`.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyHostNameKey`
A `CFString` value containing either the hostname or IP number of the proxy host.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyPasswordKey`
The password to be used when contacting the proxy. This key is only present if the password can be determined from the information passed in. (External credential stores such as the keychain are not consulted.)

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyPortNumberKey`
A `CFNumber` value specifying the port that should be used to contact the proxy.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeKey`
Specifies the type of proxy. The value can be any of the values listed in "Proxy Types" (page 144).

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyUsernameKey`
The username to be used when contacting the proxy. This key is only present if the username can be determined from the information passed in. (External credential stores such as the keychain are not consulted.)

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

Proxy Types

Constants that specify the type of proxy.

```
const CFStringRef kCFProxyTypeNone;
const CFStringRef kCFProxyTypeAutoConfigurationURL;
const CFStringRef kCFProxyTypeAutoConfigurationJavaScript;
const CFStringRef kCFProxyTypeFTP;
const CFStringRef kCFProxyTypeHTTP;
const CFStringRef kCFProxyTypeHTTPS;
const CFStringRef kCFProxyTypeSOCKS;
```

Constants

`kCFProxyTypeNone`
Specifies that no proxy should be used.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeAutoConfigurationURL`
: Specifies that the proxy is determined by an autoconfiguration file at a given URL.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeAutoConfigurationJavaScript`
: Specifies that the proxy is determined by a provided autoconfiguration script.

Available in iOS 4.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeFTP`
: Specifies an FTP proxy.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeHTTP`
: Specifies an HTTP proxy.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeHTTPS`
: Specifies an HTTPS proxy.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFProxyTypeSOCKS`
: Specifies a SOCKS proxy.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

Global Proxy Settings Constants

Constants for keys in the global proxy settings dictionary returned by `CFNetworkCopySystemProxySettings` (page 140).

```
const CFStringRef kCFNetworkProxiesHTTPEnable;
const CFStringRef kCFNetworkProxiesHTTPPort;
const CFStringRef kCFNetworkProxiesHTTPProxy;
const CFStringRef kCFNetworkProxiesProxyAutoConfigEnable;
const CFStringRef kCFNetworkProxiesProxyAutoConfigJavaScript;
const CFStringRef kCFNetworkProxiesProxyAutoConfigURLString;
```

Constants

`kCFNetworkProxiesHTTPEnable`
: Value is a `CFNumber` object indicating whether an HTTP proxy is enabled. The proxy is enabled if the key is present and the associated value is nonzero.

Available in iOS 2.0 and later.

Declared in `CFProxySupport.h`.

`kCFNetworkProxiesHTTPPort`
: Value is a `CFNumber` object containing the port number associated with the HTTP proxy.

  Available in iOS 2.0 and later.

  Declared in `CFProxySupport.h`.

`kCFNetworkProxiesHTTPProxy`
: Value is a `CFString` object containing the HTTP proxy host name or IP number.

  Available in iOS 2.0 and later.

  Declared in `CFProxySupport.h`.

`kCFNetworkProxiesProxyAutoConfigEnable`
: Value is a `CFNumber` object indicating whether proxy autoconfiguration is enabled. Proxy autoconfiguration is enabled if the key is present and the associated value is nonzero.

  Available in iOS 2.0 and later.

  Declared in `CFProxySupport.h`.

`kCFNetworkProxiesProxyAutoConfigJavaScript`
: Value is a `CFString` object that contains the full JavaScript source of the ProxyAutoConfig (PAC) file.

  Available in iOS 4.0 and later.

  Declared in `CFProxySupport.h`.

`kCFNetworkProxiesProxyAutoConfigURLString`
: Value is a `CFString` object that contains the URL of the proxy autoconfiguration (PAC) file.

  Available in iOS 2.0 and later.

  Declared in `CFProxySupport.h`.

Document Revision History

This table describes the changes to *CFNetwork Framework Reference*.

Date	Notes
2010-02-24	Added the CFHTTPStream and CFNetDiagnostics references.
2008-07-08	Updated for iOS.

Document Revision History