NSNetService Class Reference

Developer

Contents

NSNetService Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	BonjourWeb CryptoExercise SimpleNetworkStreams WiTap

Overview

The `NSNetService` class represents a network service that your application publishes or uses as a client. This class and the `NSNetServiceBrowser` class use multicast DNS to convey information about network services to and from your application. The API of `NSNetService` provides a convenient way to publish the services offered by your application and to resolve the socket address for a service.

The types of services you access using `NSNetService` are the same types that you access directly using BSD sockets. HTTP and FTP are two services commonly provided by systems. (For a list of common services and the ports used by those services, see the file `/etc/services`.) Applications can also define their own custom services to provide specific data to clients.

You can use the `NSNetService` class as either a publisher of a service or as a client of a service. If your application publishes a service, your code must acquire a port and prepare a socket to communicate with clients. Once your socket is ready, you use the `NSNetService` class to notify clients that your service is ready.

If your application is the client of a network service, you can either create an `NSNetService` object directly (if you know the exact host and port information) or you can use an `NSNetServiceBrowser` object to browse for services.

To publish a service, you must initialize your `NSNetService` object with the service name, domain, type, and port information. All of this information must be valid for the socket created by your application. Once initialized, you call the `publish` (page 12) method to broadcast your service information out to the network.

When connecting to a service, you would normally use the `NSNetServiceBrowser` class to locate the service on the network and obtain the corresponding `NSNetService` object. Once you have the object, you proceed to call the `resolveWithTimeout:` (page 13) method to verify that the service is available and ready for your application. If it is, the `addresses` (page 20) method returns the socket information you can use to connect to the service.

The methods of `NSNetService` operate asynchronously so that your application is not impacted by the speed of the network. All information about a service is returned to your application through the `NSNetService` object's delegate. You must provide a delegate object to respond to messages and to handle errors appropriately.

Tasks

Creating Network Services

– `initWithDomain:type:name:` (page 9)
  Returns the receiver, initialized as a network service of a given type and sets the initial host information.

– `initWithDomain:type:name:port:` (page 10)
  Initializes the receiver as a network service of type `type` at the socket location specified by `domain`, `name`, and `port`.

Configuring Network Services

+ `dataFromTXTRecordDictionary:` (page 7)
  Returns an `NSData` object representing a TXT record formed from a given dictionary.

+ `dictionaryFromTXTRecordData:` (page 8)
  Returns a dictionary representing a TXT record given as an `NSData` object.

– `getInputStream:outputStream:` (page 8)
  Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.

– `TXTRecordData` (page 16)
  Returns the TXT record for the receiver.

– `setTXTRecordData:` (page 14)
  Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.

– `addresses` (page 20) Available in iOS 2.0 through iOS 6.1
  Returns an array containing `NSData` objects, each of which contains a socket address for the service.

– `delegate` (page 20) Available in iOS 2.0 through iOS 6.1
  Returns the delegate for the receiver.

– `domain` (page 21) Available in iOS 2.0 through iOS 6.1
  Returns the domain name of the service.

– `hostName` (page 21) Available in iOS 2.0 through iOS 6.1
  Returns the host name of the computer providing the service.

– `name` (page 22) Available in iOS 2.0 through iOS 6.1
  Returns the name of the service.

– `setDelegate:` (page 23) Available in iOS 2.0 through iOS 6.1
  Sets the delegate for the receiver.

– `type` (page 23) Available in iOS 2.0 through iOS 6.1
  Returns the type of the service.

Managing Run Loops

– `scheduleInRunLoop:forMode:` (page 14)
  Adds the service to the specified run loop.

– `removeFromRunLoop:forMode:` (page 13)
  Removes the service from the given run loop for a given mode.

Using Network Services

– `publish` (page 12)
  Attempts to advertise the receiver's on the network.

– `publishWithOptions:` (page 12)
  Attempts to advertise the receiver on the network, with the given options.

– `resolveWithTimeout:` (page 13)
    Starts a resolve process of a finite duration for the receiver.

– `startMonitoring` (page 15)
    Starts the monitoring of TXT-record updates for the receiver.

– `stop` (page 15)
    Halts a currently running attempt to publish or resolve a service.

– `stopMonitoring` (page 16)
    Stops the monitoring of TXT-record updates for the receiver.

– `port` (page 22) Available in iOS 2.0 through iOS 6.1
    Provides the port of the receiver.

– `resolve` (page 24) Deprecated in iOS 2.0
    Starts a resolve process for the receiver. (Deprecated. Use `resolveWithTimeout:` (page 13) instead.)

Class Methods

dataFromTXTRecordDictionary:

Returns an `NSData` object representing a TXT record formed from a given dictionary.

```
+ (NSData *)dataFromTXTRecordDictionary:(NSDictionary *)txtDictionary
```

Parameters

`txtDictionary`
    A dictionary containing a TXT record.

Return Value

An `NSData` object representing TXT data formed from `txtDictionary`. Fails an assertion if `txtDictionary` cannot be represented as an `NSData` object.

Availability

Available in iOS 2.0 and later.

See Also

– `TXTRecordData` (page 16)
+ `dictionaryFromTXTRecordData:` (page 8)

Declared in

`NSNetServices.h`

dictionaryFromTXTRecordData:

Returns a dictionary representing a TXT record given as an `NSData` object.

```
+ (NSDictionary *)dictionaryFromTXTRecordData:(NSData *)txtData
```

Parameters

`txtData`

A data object encoding a TXT record.

Return Value

A dictionary representing `txtData`. The dictionary's keys are `NSString` objects using UTF8 encoding. The values associated with all the dictionary's keys are `NSData` objects that encapsulate strings or data.

Fails an assertion if `txtData` cannot be represented as an `NSDictionary` object.

Availability

Available in iOS 2.0 and later.

See Also

`– TXTRecordData` (page 16)
`+ dataFromTXTRecordDictionary:` (page 7)

Related Sample Code
BonjourWeb

Declared in

`NSNetServices.h`

Instance Methods

getInputStream:outputStream:

Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.

```
– (BOOL)getInputStream:(NSInputStream **)inputStream  outputStream:(NSOutputStream
**)outputStream
```

Parameters

`inputStream`

Upon return, the input stream for the receiver.

`outputStream`
: Upon return, the output stream for the receiver.

Return Value

`YES` if the streams are created successfully, otherwise `NO`.

Discussion

After this method is called, no delegate callbacks are called by the receiver.

Availability

Available in iOS 2.0 and later.

Related Sample Code
WiTap

Declared in

`NSNetServices.h`

initWithDomain:type:name:

Returns the receiver, initialized as a network service of a given type and sets the initial host information.

```
- (id)initWithDomain:(NSString *)domain type:(NSString *)type name:(NSString *)name
```

Parameters

`domain`
: The domain for the service. For the local domain, use `@"local."` not `@""`.

`type`
: The network service type.

  `type` must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string "`_http._tcp.`". Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

`name`
: The name of the service to resolve.

Return Value

The receiver, initialized as a network service named `name` of type `type` in the domain `domain`.

Discussion

This method is the appropriate initializer to use to resolve a service—to publish a service, use initWithDomain:type:name:port: (page 10).

If you know the values for `domain`, `type`, and `name` of the service you wish to connect to, you can create an `NSNetService` object using this initializer and call `resolveWithTimeout:` (page 13) on the result.

You cannot use this initializer to publish a service. This initializer passes an invalid port number to the designated initializer, which prevents the service from being registered. Calling `publish` (page 12) on an `NSNetService` object initialized with this method generates a call to your delegate's `netService:didNotPublish:` method with an `NSNetServicesBadArgumentError` (page 18) error.

Availability

Available in iOS 2.0 and later.

See Also

– `initWithDomain:type:name:port:` (page 10)

Related Sample Code

SimpleNetworkStreams

WiTap

Declared in

`NSNetServices.h`

initWithDomain:type:name:port:

Initializes the receiver as a network service of type `type` at the socket location specified by `domain`, `name`, and `port`.

```
- (id)initWithDomain:(NSString *)domain  type:(NSString *)type  name:(NSString *)name
  port:(int)port
```

Parameters

`domain`

The domain for the service. For the local domain, use `@"local."` not `@""`.

It is generally preferred to use a `NSNetServiceBrowser` object to obtain the local registration domain in which to publish your service. To use this default domain, simply pass in an empty string (`@""`).

`type`

The network service type.

`type` must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character (“_”). For example, to search for an HTTP service on TCP, you would use the type string "`_http._tcp.`". Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

`name`

The name by which the service is identified to the network. The name must be unique. If you pass the empty string (`@""`), the system automatically advertises your service using the computer name as the service name.

`port`

The port on which the service is published.

`port` must be a port number acquired by your application for the service.

Discussion

You use this method to create a service that you wish to publish on the network. Although you can also use this method to create a service you wish to resolve on the network, it is generally more appropriate to use the `initWithDomain:type:name:` (page 9) method instead.

When publishing a service, you must provide valid arguments in order to advertise your service correctly. If the host computer has access to multiple registration domains, you must create separate `NSNetService` objects for each domain. If you attempt to publish in a domain for which you do not have registration authority, your request may be denied.

It is acceptable to use an empty string for the `domain` argument when publishing or browsing a service, but do not rely on this for resolution.

This method is the designated initializer.

Availability

Available in iOS 2.0 and later.

See Also

– `initWithDomain:type:name:` (page 9)

Related Sample Code

CryptoExercise

SimpleNetworkStreams

WiTap

Declared in
`NSNetServices.h`

publish

Attempts to advertise the receiver's on the network.

```
- (void)publish
```

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

Availability
Available in iOS 2.0 and later.

See Also
– `stop` (page 15)

Declared in
`NSNetServices.h`

publishWithOptions:

Attempts to advertise the receiver on the network, with the given options.

```
- (void)publishWithOptions:(NSNetServiceOptions)serviceOptions
```

Parameters
`serviceOptions`
Options for the receiver.

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

Availability
Available in iOS 2.0 and later.

Declared in
`NSNetServices.h`

removeFromRunLoop:forMode:

Removes the service from the given run loop for a given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)aRunLoop  forMode:(NSString *)mode
```

Parameters

`aRunLoop`

The run loop from which to remove the receiver.

`mode`

The run loop mode from which to remove the receiver. Possible values for `mode` are discussed in the "Constants" section of `NSRunLoop`.

Discussion

You can use this method in conjunction with `scheduleInRunLoop:forMode:` (page 14) to transfer the service to a different run loop. Although it is possible to remove an `NSNetService` object completely from any run loop and then attempt actions on it, it is an error to do so.

Availability

Available in iOS 2.0 and later.

See Also

– `scheduleInRunLoop:forMode:` (page 14)

Declared in

`NSNetServices.h`

resolveWithTimeout:

Starts a resolve process of a finite duration for the receiver.

```
- (void)resolveWithTimeout:(NSTimeInterval)timeout
```

Parameters

`timeout`

The maximum number of seconds to attempt a resolve. A value of 0.0 indicates no timeout and a resolve process of indefinite duration.

Discussion

If the resolve succeeds before the `timeout` period lapses, the receiver sends `netServiceDidResolveAddress:` to the delegate. Otherwise, the receiver sends `netService:didNotResolve:` to the delegate.

Availability
Available in iOS 2.0 and later.

See Also
– `addresses` (page 20)
– `stop` (page 15)

Declared in
`NSNetServices.h`

scheduleInRunLoop:forMode:

Adds the service to the specified run loop.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

`aRunLoop`
The run loop to which to add the receiver.

`mode`
The run loop mode to which to add the receiver. Possible values for `mode` are discussed in the "Constants" section of `NSRunLoop`.

Discussion
You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 13) to transfer a service to a different run loop. You should not attempt to run a service on multiple run loops.

Availability
Available in iOS 2.0 and later.

See Also
– `removeFromRunLoop:forMode:` (page 13)

Declared in
`NSNetServices.h`

setTXTRecordData:

Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.

```
- (BOOL)setTXTRecordData:(NSData *)recordData
```

Parameters

`recordData`

The TXT record for the receiver.

Return Value

`YES` if `recordData` is successfully set as the TXT record, otherwise `NO`.

Availability

Available in iOS 2.0 and later.

See Also

– `TXTRecordData` (page 16)

Declared in

`NSNetServices.h`

startMonitoring

Starts the monitoring of TXT-record updates for the receiver.

```
- (void)startMonitoring
```

Discussion

The delegate must implement `netService:didUpdateTXTRecordData:`, which is called when the TXT record for the receiver is updated.

Availability

Available in iOS 2.0 and later.

See Also

– `stopMonitoring` (page 16)

Declared in

`NSNetServices.h`

stop

Halts a currently running attempt to publish or resolve a service.

```
- (void)stop
```

Discussion

This method results in the sending of a `netServiceDidStop:` message to the delegate.

Availability

Available in iOS 2.0 and later.

Declared in

`NSNetServices.h`

stopMonitoring

Stops the monitoring of TXT-record updates for the receiver.

```
- (void)stopMonitoring
```

Availability

Available in iOS 2.0 and later.

See Also

`– startMonitoring` (page 15)

Declared in

`NSNetServices.h`

TXTRecordData

Returns the TXT record for the receiver.

```
- (NSData *)TXTRecordData
```

Availability

Available in iOS 2.0 and later.

See Also

`– setTXTRecordData:` (page 14)
`+ dictionaryFromTXTRecordData:` (page 8)
`+ dataFromTXTRecordDictionary:` (page 7)

Related Sample Code
BonjourWeb

Declared in

`NSNetServices.h`

Constants

NSNetServices Errors

If an error occurs, the delegate error-handling methods return a dictionary with the following keys.

```
extern NSString *NSNetServicesErrorCode;
extern NSString *NSNetServicesErrorDomain;
```

Constants

`NSNetServicesErrorCode`

This key identifies the error that occurred during the most recent operation.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesErrorDomain`

This key identifies the originator of the error, which is either the `NSNetService` object or the mach network layer. For most errors, you should not need the value provided by this key.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

Declared in

`NSNetServices.h`

NSNetServicesError

These constants identify errors that can occur when accessing net services.

```
typedef enum {
   NSNetServicesUnknownError = -72000,
   NSNetServicesCollisionError = -72001,
   NSNetServicesNotFoundError    = -72002,
   NSNetServicesActivityInProgress = -72003,
   NSNetServicesBadArgumentError = -72004,
   NSNetServicesCancelledError = -72005,
   NSNetServicesInvalidError = -72006,
   NSNetServicesTimeoutError = -72007,
} NSNetServicesError;
```

Constants

`NSNetServicesUnknownError`

An unknown error occurred.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesCollisionError`

The service could not be published because the name is already in use. The name could be in use locally or on another system.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesNotFoundError`

The service could not be found on the network.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesActivityInProgress`

The net service cannot process the request at this time. No additional information about the network state is known.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesBadArgumentError`

An invalid argument was used when creating the `NSNetService` object.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesCancelledError`

The client canceled the action.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesInvalidError`

The net service was improperly configured.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesTimeoutError`

The net service has timed out.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

Declared in
`NSNetServices.h`

NSNetServiceOptions

These constants specify options for a network service.

```
enum {
   NSNetServiceNoAutoRename = 1 << 0
};
typedef NSUInteger NSNetServiceOptions;
```

Constants
`NSNetServiceNoAutoRename`

Specifies that the network service not rename itself in the event of a name collision.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

Availability
Available in iOS 2.0 and later.

Declared in
`NSNetServices.h`

Deprecated NSNetService Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Available in iOS 2.0 through iOS 6.1

addresses

Returns an array containing `NSData` objects, each of which contains a socket address for the service. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSArray *)addresses
```

Return Value
An array containing `NSData` objects, each of which contains a socket address for the service. Each `NSData` object in the returned array contains an appropriate `sockaddr` structure that you can use to connect to the socket. The exact type of this structure depends on the service to which you are connecting. If no addresses were resolved for the service, the returned array contains zero elements.

Discussion
It is possible for a single service to resolve to more than one address or not resolve to any addresses. A service might resolve to multiple addresses if the computer publishing the service is currently multihoming.

Availability
Available in iOS 2.0 through iOS 6.1.

See Also
`– resolve` (page 24)

Declared in
`NSNetServices.h`

delegate

Returns the delegate for the receiver. (Available in iOS 2.0 through iOS 6.1.)

```
- (id < NSNetServiceDelegate >)delegate
```

Return Value
The delegate for the receiver.

Availability
Available in iOS 2.0 through iOS 6.1.

See Also
– `setDelegate:` (page 23)

Related Sample Code
CryptoExercise

WiTap

Declared in
`NSNetServices.h`

domain

Returns the domain name of the service. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSString *)domain
```

Return Value
The domain name of the service.

This can be an explicit domain name or it can contain the generic local domain name, `@"local."` (note the trailing period, which indicates an absolute name).

Availability
Available in iOS 2.0 through iOS 6.1.

Related Sample Code
SimpleNetworkStreams

Declared in
`NSNetServices.h`

hostName

Returns the host name of the computer providing the service. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSString *)hostName
```

Return Value

The host name of the computer providing the service. Returns `nil` if a successful resolve has not occurred.

Availability

Available in iOS 2.0 through iOS 6.1.

Related Sample Code
BonjourWeb

Declared in

`NSNetServices.h`

name

Returns the name of the service. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSString *)name
```

Return Value

The name of the service.

Availability

Available in iOS 2.0 through iOS 6.1.

Related Sample Code
BonjourWeb

SimpleNetworkStreams

WiTap

Declared in

`NSNetServices.h`

port

Provides the port of the receiver. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSInteger)port
```

Return Value

The receiver's port. `-1` when it has not been resolved.

Availability

Available in iOS 2.0 through iOS 6.1.

Related Sample Code
BonjourWeb

Declared in
`NSNetServices.h`

setDelegate:

Sets the delegate for the receiver. (Available in iOS 2.0 through iOS 6.1.)

```
- (void)setDelegate:(id < NSNetServiceDelegate >)delegate
```

Parameters
`delegate`
The delegate for the receiver. The delegate must conform to the `NSNetServiceDelegate Protocol` protocol.

Discussion
The delegate is not retained.

Availability
Available in iOS 2.0 through iOS 6.1.

See Also
– `delegate` (page 20)

Declared in
`NSNetServices.h`

type

Returns the type of the service. (Available in iOS 2.0 through iOS 6.1.)

```
- (NSString *)type
```

Return Value
The type of the service.

Availability
Available in iOS 2.0 through iOS 6.1.

Related Sample Code
SimpleNetworkStreams

Declared in
`NSNetServices.h`

Deprecated in iOS 2.0

resolve

Starts a resolve process for the receiver. (Deprecated in iOS 2.0. Use `resolveWithTimeout:` (page 13) instead.)

```
- (void)resolve
```

Discussion
Attempts to determine at least one address for the receiver. This method returns immediately, with success or failure indicated by the callbacks to the delegate.

In Mac OS X v10.4, this method calls `resolveWithTimeout:` (page 13) with a timeout value of `5`.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also
– `addresses` (page 20)
– `stop` (page 15)
– `resolveWithTimeout:` (page 13)

Declared in
`NSNetServices.h`

Document Revision History

This table describes the changes to *NSNetService Class Reference* .

Date	Notes
2011-01-04	Noted that a timeout value of 0.0 means to attempt resolution indefinitely. Added note that an empty name argument in initWithDomain:type:name: results in using the computer's name as the name of the service.
2010-03-22	Updated for iOS 4.0. Delegate methods moved to NSNetServiceDelegate Protocol Reference.
2009-04-08	Miscellaneous edits.
2008-02-08	Corrected typographical errors.
2007-04-24	Updated for OS X v10.5.
2006-05-23	Added "NSNetServices and CFNetServices Programming Guide" as a companion document. First publication of this content as a separate document.