NSNetServices and CFNetServices Programming Guide

Networking & Internet: Services & Discovery

2010-03-24

Apple Inc.
© 2010 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Bonjour, Carbon, Cocoa, iPhone, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to NSNetServices and CFNetServices Programming Guide 7

Who Should Read This Document 7
Organization of This Document 7

Foundation Network Services Architecture 9

Foundation Classes for Network Services 9
  NSNetService 9
  NSNetServiceBrowser 9
  Asynchronous Results and Delegate Objects 10
Operations on Network Services 11
  Publication 11
  Service Discovery 12
  Resolution 13
  Browsing for Domains 14

Publishing Network Services 15

The Publication Process 15
Setting Up Socket Ports 16
Initializing and Publishing a Network Service 16
Implementing Delegate Methods for Publication 18

Browsing for Network Services 21

The Browsing Process 21
Initializing the Browser and Starting a Search 21
Implementing Delegate Methods for Browsing 22

Browsing for Domains 27

About Domain Browsing 27
Initializing the Browser and Starting a Search 28
Implementing Delegate Methods for Browsing 28

Resolving and Using Network Services 33

The Resolution Process 33
Obtaining and Resolving an NSNetService Object 33
Implementing Delegate Methods for Resolution 34
Connecting to a Network Service 36

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Monitoring a Service 37

CFNetServices 39

Requirements 39
Using the CFNetServices API 39
Publishing a Service 39
  Creating a CFNetService 39
  Registering a CFNetService 40
Browsing for Services 41
Resolving a Service 42
Monitoring a Service 43
Asynchronous and Synchronous Modes 43
Shutting Down Services and Searches 45

Document Revision History 47

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Figures, Tables, and Listings

Foundation Network Services Architecture 9

Figure 1 Asynchronous service resolution returning results to a delegate object 10
Figure 2 Service publication with NSNetService 12
Figure 3 Service discovery with NSNetServiceBrowser 13
Figure 4 Service resolution with NSNetService 14

Publishing Network Services 15

Listing 1 Initializing and publishing a Bonjour network service 17
Listing 2 Interface for an NSNetService delegate object (publication) 19
Listing 3 Implementation for an NSNetService delegate object (publication) 19

Browsing for Network Services 21

Listing 1 Browsing for Bonjour network services 22
Listing 2 Interface for an NSNetServiceBrowser delegate object (services) 23
Listing 3 Implementation for an NSNetServiceBrowser delegate object (services) 23

Browsing for Domains 27

Listing 1 Browsing for registration domains 28
Listing 2 Interface for an NSNetServiceBrowser delegate object (domains) 29
Listing 3 Implementation for an NSNetServiceBrowser delegate object (domains) 29

Resolving and Using Network Services 33

Listing 1 Resolving network services with NSNetService 34
Listing 2 Interface for an NSNetService delegate object (resolution) 35
Listing 3 Implementation for an NSNetService delegate object (resolution) 35
Listing 4 Connecting to a resolved Bonjour network service 36

CFNetServices 39

Table 1 Behavior of certain CFNetServices functions in asynchronous and synchronous mode 44
Listing 1 Creating a CFNetService 40
Listing 2 Registering an Asynchronous Service 40
Listing 3 Browsing Asynchronously for Services 41
Listing 4 Resolving a Service Asynchronously 42
Listing 5 Canceling an Asynchronous CFNetService Resolve Process 45
Listing 6 Stop Browsing for Services 45

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Introduction to NSNetServices and CFNetServices Programming Guide

NSNetServices and CFNetServices Programming Guide is a collection of articles describing the APIs that implement Bonjour on the Foundation and Core Foundation levels. The articles describe how to use NSNetServices, and CFNetServices for tasks such as publishing a service and browsing for services.

Who Should Read This Document

This document is intended for developers who wish to add Bonjour functionality to their Cocoa or Carbon application. It assumes that the developer already is familiar with the basics of Bonjour from reading *Bonjour Overview*. If developers want to add Bonjour functionality into a non-Cocoa, or a BSD-style application, it is recommended that they investigate the *DNS Service Discovery Programming Guide* first.

Organization of This Document

This document contains the following articles:

- "Foundation Network Services Architecture" (page 9) describes the functions available in the NSNetServices framework.
- "Publishing Network Services" (page 15) explains how to publish a Bonjour service.
- "Browsing for Network Services" (page 21) describes how to search for available Bonjour services.
- "Browsing for Domains" (page 27) explains how to browse for Bonjour domains
- "Resolving and Using Network Services" (page 33) explains how to resolve a Bonjour service.
- "CFNetServices" (page 39) explains how to perform all of the above tasks using the Core Foundation framework.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Foundation Network Services Architecture

This article describes the structure of the Foundation classes used for network services, and how the methods in these classes operate.

Foundation Classes for Network Services

The Foundation framework defines two classes for managing Bonjour network services. These classes correspond to the two basic elements of network services: services and service browsers. NSNetService represents an actual instance of a service, and provides methods both for publishing a local service to the network and for connecting to a remote service. NSNetServiceBrowser acts as a browser for a particular type of service; it queries the network for available services and collects the results.

NSNetService

The NSNetService class represents a single instance of a service. The service can either be a remote service that your application wants to use, or a local service your application is publishing; however, it is never both. NSNetService instances perform all operations asynchronously, returning results to a delegate object for processing.

NSNetService objects that represent remote services are initialized with the service name, type, and domain, but no host name, IP address or port number. The name, type, and domain are used to resolve the instance into socket information (IP address and port number) so your application can connect to the service.

NSNetService objects used for publishing local services to the network are initialized with the service name, type, and domain, and also the service's port number. They use this information to announce the necessary information to the network through the multicast DNS responder.

NSNetServiceBrowser

An NSNetServiceBrowser object represents one of two things: A browser for a single type of service, or a browser for domains. Like NSNetService, NSNetServiceBrowser instances perform all operations asynchronously, returning results to a delegate object for processing. At any given time, a single NSNetServiceBrowser object can execute at most one search operation; if you need to search for multiple types of services at once, use multiple NSNetServiceBrowser objects.

Most of the time, you use NSNetServiceBrowser to search for a specific type of service. For example, you might set up a service browser to search for FTP services available on the local network and present the list of services to the user, letting the user connect to one of them. The results of such a search are instances of NSNetService corresponding to each remote service.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

You can also use NSNetServiceBrowser to search for available domains. By passing NSNetServiceBrowser the empty string (`@""`) for the domain, it will search for all potential domains. However, just because you have access to all available domains, does not mean that you should begin searching through them sequentially. It is best to present the user with the list of domains, allowing him to select one which can then be scanned for services.

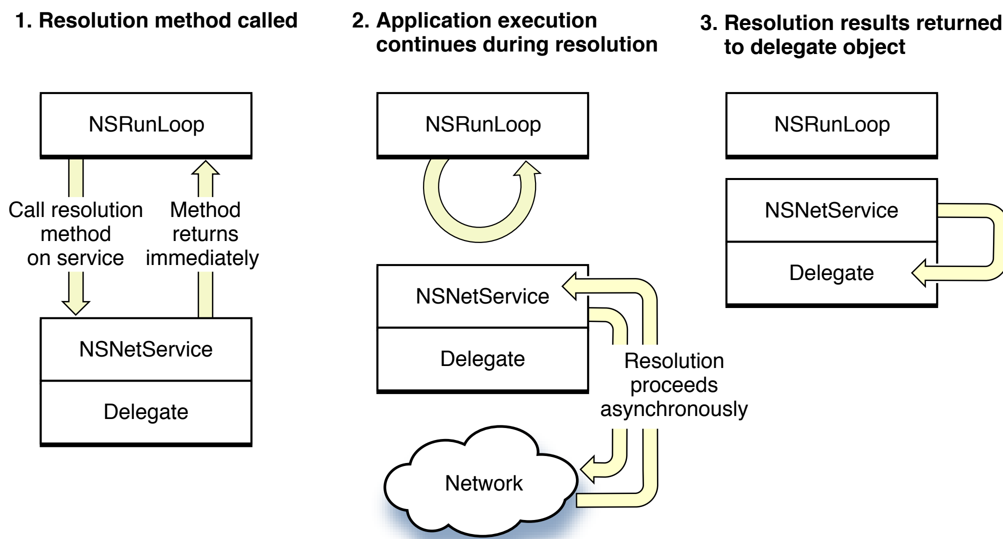
Asynchronous Results and Delegate Objects

Due to the length of time it takes for network discovery, the NSNetService and NSNetServiceBrowser APIs perform all their operations asynchronously. The methods provided by NSNetService and NSNetServiceBrowser return immediately, so your application can continue executing while network operations take place. However, your application still needs to process the information returned by the objects. Instead of requiring you to subclass NSNetService and NSNetServiceBrowser to handle results, both classes send the results of network operations to delegate objects that implements the appropriate methods to handle the results.

A common network operation is resolving a service instance name—such as `3rd Floor Copy Room._printer._tcp.local.`—into socket information (IP address and port number). Figure 1 illustrates how this resolution process happens asynchronously and returns results to the delegate object.

At some point (step 1), a message is sent to the NSNetService object requesting that it retrieve socket addresses for the service; this method returns immediately. In step 2, the application continues execution as the resolution proceeds. At some later point in time, the resolution finishes (step 3), and the NSNetService object returns results to its delegate object.

Figure 1 Asynchronous service resolution returning results to a delegate object



2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Operations on Network Services

The architecture for Bonjour network services in the Foundation framework abstracts your application away from the details of DNS record management, and lets you manage services in terms of the three fundamental operations defined for Bonjour network services:

- Publication (advertising a service)
- Service discovery (browsing for available services)
- Resolution (translating service names to addresses and port numbers for use)

In Cocoa applications, you perform these operations using NSNetService and NSNetServiceBrowser objects. NSNetService objects represent instances of Bonjour net services, and are involved in all three operations. The NSNetServiceBrowser class, as its name suggests, defines an interface for service discovery; it also lets you search for domains available for publication and browsing.

The relationship between these operations and the two classes is discussed in the following sections.

Publication

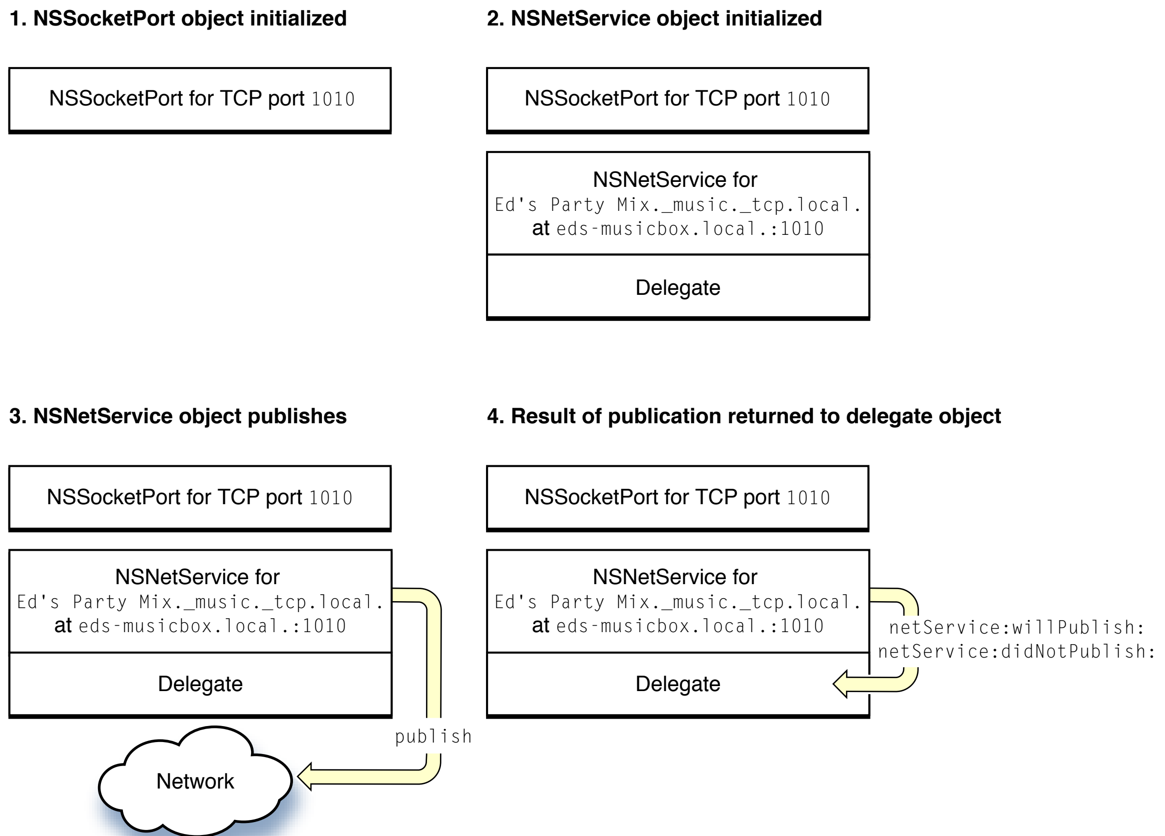
The NSNetService class handles service publication, as illustrated in Figure 2. To publish a service, an application first sets up a valid socket for communication, for example with NSSocketPort (step 1). Once the socket is ready, the application initializes an NSNetService object with the socket and domain information (step 2), and sets up a delegate object to receive results. The object automatically registers itself with the current run loop. In step 3, the application sends a message to the NSNetService object requesting that the service be published to the network, and publication proceeds asynchronously. In step 4, the NSNetService object returns results to your delegate.

When the service is about to be published, the delegate is notified. If publication fails for any reason, the delegate is also notified, along with appropriate error information. If publication proceeds successfully, no further messages are sent to the delegate.

For step-by-step instructions and code examples about service publication, see "Publishing Network Services" (page 15).

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Figure 2 Service publication with NSNetService

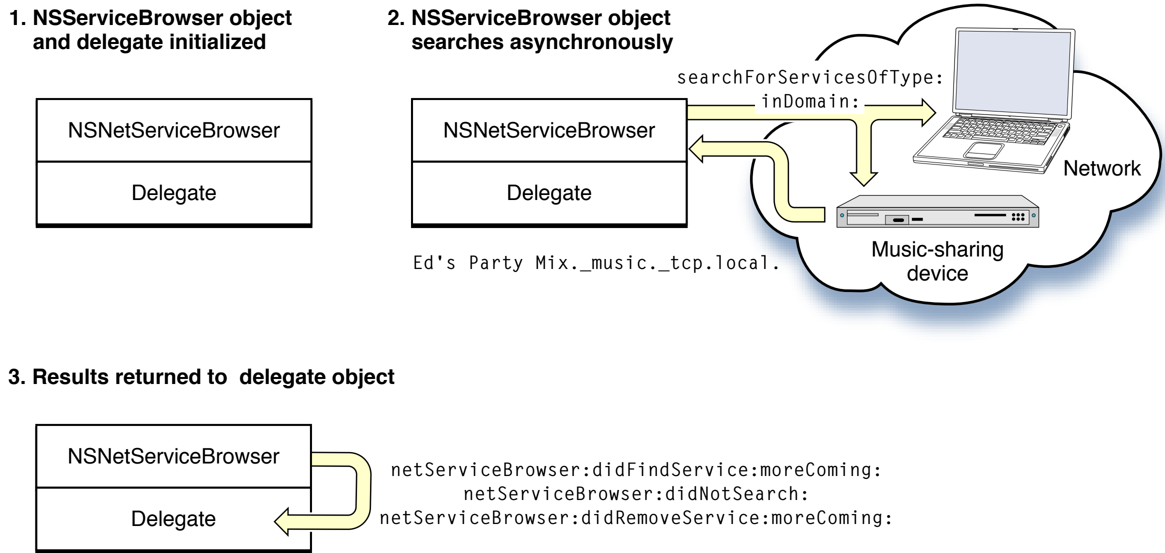


Service Discovery

To discover services advertised on the network, you use the NSNetServiceBrowser class, as shown in Figure 3. In step 1, the application initializes an NSNetServiceBrowser object and associates a delegate object to it. In step 2, the NSNetServiceBrowser object searches asynchronously for services. In step 3, results are returned to the delegate object. The last step can happen a number of times as new services are found on the network. A browser can search for an extended period of time, and the delegate is notified as new services come online or shut down.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Figure 3 Service discovery with NSNetServiceBrowser



For step-by-step instructions and code examples about service discovery, see "Browsing for Network Services" (page 21).

Resolution

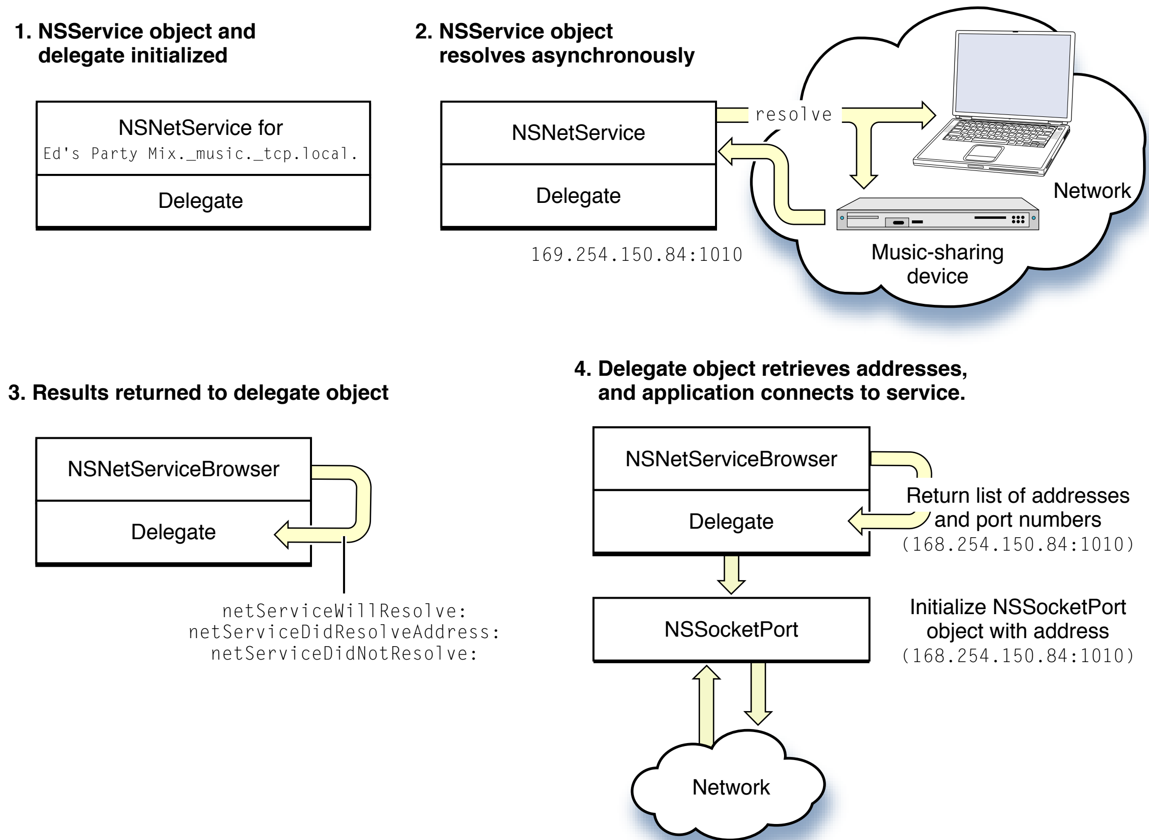
If you know the name, type, and domain of a service, you can initialize an NSNetService object with this information, and ask the object to resolve the service name into socket addresses. To prevent your application from slowing down, resolution also takes place asynchronously, returning results or error messages to the NSNetService object's delegate. If valid addresses were found for the service, your application can then use the addresses to make a socket connection.

Figure 4 illustrates this process. In step 1, the application initializes an NSNetService instance for the service, in this case a local `music` service over TCP called `Ed's Party Mix`. In step 2, the NSNetService object receives a `resolve` message. The resolution proceeds asynchronously, and at some point it receives an IP address and port number for the service (`169.254.150.84:1010`). In step 3, the delegate is notified, and in step 4, the delegate asks the NSNetService object for a list of addresses and port numbers, in this case just one of each.

For step-by-step instructions and code examples about service resolution, see "Resolving and Using Network Services" (page 33).

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Figure 4 Service resolution with NSNetService



Browsing for Domains

NSNetServiceBrowser objects can also search for domains in much the same way as they search for services. Instead of returning NSNetService objects to delegate objects, they return domain names.

For step-by-step instructions and code examples about domain browsing, see "Browsing for Domains" (page 27)

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Publishing Network Services

Bonjour enables dynamic discovery of network services on IP networks without a centralized directory server. The Foundation framework's NSNetService class represents instances of Bonjour network services. This article describes the process for publishing Bonjour network services with NSNetService.

The Publication Process

Bonjour network services use standard DNS information to advertise their existence to potential clients on a network. In Cocoa, the NSNetService class handles the details of service publication.

Typically, you use NSNetService to publish a service provided by a socket owned by the same process. This is not a requirement, so you can use the class to advertise on behalf of another process's service—for example, an FTP server process that has not yet been updated to support Bonjour. However, if you are creating an IP network service, you should include Bonjour publication code as part of its startup process.

Because network activity can sometimes take some time, NSNetService objects process publication requests asynchronously, delivering information through delegate methods. To use NSNetService correctly, your application must assign a delegate to each NSNetService instance it creates. Because the identity of the NSNetService object is passed as a parameter in delegate methods, you can use one delegate for multiple NSNetService objects.

Publishing a Bonjour network service takes four steps:

1. Set up a valid socket port for communication.
2. Initialize an NSNetService instance with name, type, domain and socket information, and assign a delegate to the object.
3. Publish the NSNetService instance.
4. Respond to messages sent to the NSNetService object's delegate.

The following sections describe these steps in detail.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

> **Note:** Bonjour does not need its own socket, only the network service does. If you are making an existing network application Bonjour enabled and already have socket ports configured for the network services, you can ignore step number one.

Setting Up Socket Ports

Bonjour network services are set up as standard TCP or UDP socket ports over IP. Mac OS X provides a number of APIs to help you manage sockets:

- CFNetwork, part of the Core Services framework, for HTTP sockets
- NSSocketPort, part of the Foundation framework
- CFStream and CFSocket
- Raw BSD sockets, using the API provided in `<sys/socket.h>` and `<netinet/in.h>`.

Using NSSocketPort or CFStream and CFSocket generally means also interacting with the lower-level BSD sockets layer. NSSocketPort objects can also be used as endpoints for Cocoa distributed objects communication advertised as a Bonjour network service. CFNetwork also provides an abstraction over low-level sockets, but only for HTTP transactions.

For more detailed information, see the *CFNetwork Programming Guide*, the documentation for NSSocketPort and the many available web and print resources about BSD sockets.

Initializing and Publishing a Network Service

To initialize an NSNetService instance for publication, use the `initWithDomain:type:name:port:` method. This method sets up the instance with appropriate socket information and adds it to the current run loop.

The service type expresses both the application protocol (FTP, HTTP, and so on) and the transport protocol (TCP or UDP). The format is as described in Domain Naming Conventions, for example, `_printer._tcp` for a printer over TCP.

The service name can be any NSString. This is the name that should be presented to users, so it should be human-readable and descriptive of the specific service instance. You should let the user override any default name you provide.

It is recommended that you use the computer name as the service name. If you pass the empty string (`@""`) for the service name parameter, the system automatically advertises your service using the computer name as the service name. Another possibility is to retrieve the computer name to concatenate it with another string. In Mac OS X on the desktop, calling the `SCDynamicStore` function from the System Configuration framework returns the computer name so you can manipulate it like any other string. In iOS, you can obtain the same information from the `name` property of the `UIDevice` class.

If you want to use a run loop other than the current run loop, you can call the `removeFromRunLoop:forMode:` and `scheduleInRunLoop:forMode:` methods. The object must be scheduled in a run loop to operate.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Once the initialization is complete and valid, assign a delegate to the NSNetService object with the `setDelegate:` method. Finally, publish the service with the `publish` method, which returns immediately. It performs publication asynchronously and returns results through delegate methods.

Listing 1 demonstrates the initialization and publication process for Bonjour network services. An explanation of the code follows it. Also see the PictureSharing application in `/Developer/Examples/Foundation/PictureSharing` for a good example of service publication.

> **Note:** When the PictureSharing example code in Mac OS X v10.2 was finalized, the link-local suffix was `local.arpa.`, but it changed to `local.` before the operating system shipped. Use `local.` to specify the link-local network.

Listing 1 Initializing and publishing a Bonjour network service

```
#import <netinet/in.h>
#import <sys/socket.h>

// ...

id delegateObject;      // Assume this exists.
NSSocketPort *socket;
NSNetService *service;
struct sockaddr *addr;
int port;

socket = [[NSSocketPort alloc] init];                                   // 1
if(socket)
{
    [self setUpSocket:socket];                                          // 2

    addr = (struct sockaddr *)[[socket address] bytes];                 // 3
    if(addr->sa_family == AF_INET)
    {
        port = ntohs(((struct sockaddr_in *)addr)->sin_port);
    }
    else if(addr->sa_family == AF_INET6)
    {
        port = ntohs(((struct sockaddr_in6 *)addr)->sin6_port);
    }
    else
    {
        [socket release];
        socket = nil;
        NSLog(@"The family is neither IPv4 nor IPv6. Can't handle.");
    }
}
else
{
    NSLog(@"An error occurred initializing the NSSocketPort object.");
}

if(socket)
{
    service = [[NSNetService alloc] initWithDomain:@""                  // 4
                                    type:@"_music._tcp"
                                    name:@"" port:port];
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
    if(service)
    {
        [service setDelegate:delegateObject];                    // 5
        [service publish];                                       // 6
    }
    else
    {
        NSLog(@"An error occurred initializing the NSNetService object.");
    }
}
else
{
    NSLog(@"An error occurred initializing the NSSocketPort object.");
}
```

Here's what the code does:

1. Initializes an NSSocketPort instance with the `init` method, which sets up an available TCP/IP socket port.

2. Calls a setup method defined elsewhere in this object. This method might connect the socket to a custom BSD sockets-based service, a distributed objects connection, or other TCP/IP service.

3. Extracts the port number from the `sockaddr` structure returned by the NSSocketPort `address` method. Finds out whether the structure is IPv4-based or IPv6-based, and extracts the appropriate part of the structure.

4. If the socket initialization succeeds, initializes the NSNetService object. This example uses the default domain(s) for publication and a hypothetical TCP/IP music service.

5. Sets the delegate for the NSNetService object. This object handles all results from the NSNetService object, as described in "Implementing Delegate Methods for Publication" (page 18).

6. Finally, publishes the service to the network.

To stop a service that is already running or in the process of starting up, use the `stop` method.

Implementing Delegate Methods for Publication

NSNetService returns publication results to its delegate. If you are publishing a service, your delegate object should implement the following methods:

- `netServiceWillPublish:`
- `netService:didNotPublish:`
- `netServiceDidStop:`

The `netServiceWillPublish:` method notifies the delegate that the network is ready to publish the service. When this method is called, the service is not yet visible to the network, and publication may still fail. After this method is called, however, you can assume the service is visible unless you receive an error message.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

The `netService:didNotPublish:` method is called when publication fails for any reason. Publication can fail even after the `netServiceWillPublish:` method is called. If the delegate receives a `netService:didNotPublish:` message, you should extract the type of error from the returned dictionary using the `NSNetServicesErrorCode` key and handle the error accordingly.

One common error is `NSNetServicesCollisionError`, which is received when the service name is already in use. If your application receives this error, it should inform the user and ask for a different name. See NSNetService for a complete list of possible errors.

The `netServiceDidStop:` method gets called as a result of the `stop` message being sent to the NSNetService object. If this method gets called, the service is no longer running.

Listing 2 shows the interface for a class that acts as a delegate for multiple NSNetService objects, and Listing 3 shows its implementation. You can use this code as a starting point for more sophisticated tracking of published services.

Listing 2 Interface for an NSNetService delegate object (publication)

```
#import <Foundation/Foundation.h>

@interface NetServicePublicationDelegate : NSObject
{
    // Keeps track of active services or services about to be published
    NSMutableArray *services;
}

// NSNetService delegate methods for publication
- (void)netServiceWillPublish:(NSNetService *)netService;
- (void)netService:(NSNetService *)netService
        didNotPublish:(NSDictionary *)errorDict;
- (void)netServiceDidStop:(NSNetService *)netService;

// Other methods
- (void)handleError:(NSNumber *)error withService:(NSNetService *)service;

@end
```

Listing 3 Implementation for an NSNetService delegate object (publication)

```
#import "NetServicePublicationDelegate.h"

@implementation NetServicePublicationDelegate

- (id)init
{
    self = [super init];
    if (self) {
        services = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void)dealloc
{
    [services release];
    [super dealloc];
}
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
// Sent when the service is about to publish
- (void)netServiceWillPublish:(NSNetService *)netService
{
    [services addObject:netService];
    // You may want to do something here, such as updating a user interface
}

// Sent if publication fails
- (void)netService:(NSNetService *)netService
        didNotPublish:(NSDictionary *)errorDict
{
    [self handleError:[errorDict objectForKey:NSNetServicesErrorCode]
withService:netService];
    [services removeObject:netService];
}

// Sent when the service stops
- (void)netServiceDidStop:(NSNetService *)netService
{
    [services removeObject:netService];
    // You may want to do something here, such as updating a user interface
}

// Error handling code
- (void)handleError:(NSNumber *)error withService:(NSNetService *)service
{
    NSLog(@"An error occurred with service %@.%@.%@, error code = %@",
        [service name], [service type], [service domain], error);
    // Handle error here
}

@end
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Browsing for Network Services

An important part of Bonjour is the ability to browse for services on the network. This article describes how to use the NSNetServiceBrowser class to discover Bonjour network services.

The Browsing Process

The NSNetServiceBrowser class provides methods for browsing for available Bonjour network services.

Because of the possible delays associated with network traffic, NSNetServiceBrowser objects perform browsing asynchronously by registering with the default run loop. Browsing results are returned to your application through delegate methods. To handle results from an NSNetServiceBrowser object, you must assign it a delegate.

Browsing for Bonjour network services takes three steps:

1. Initialize an NSNetServiceBrowser instance and assign a delegate to the object.
2. Begin a search for services of a specific type in a given domain.
3. Handle search results and other messages sent to the delegate object.

The following sections describe these steps in detail.

Initializing the Browser and Starting a Search

To initialize an NSNetServiceBrowser object, use the `init` method. This sets up the browser and adds it to the current run loop. If you want to use a run loop other than the current one, use the `removeFromRunLoop:forMode:` and `scheduleInRunLoop:forMode:` methods.

To begin the browsing process, use the `searchForServicesOfType:inDomain:` method.

The service type expresses both the application protocol (FTP, HTTP, and so on.) and the transport protocol (TCP or UDP). The format is as described in Domain Naming Conventions, for example, `_printer._tcp` for an LPR printer over TCP.

The domain parameter specifies the DNS domain in which the browser performs its search. Unless you only want to browse a specific domain, pass the empty string (@"") as the domain to allow `searchForServicesOfType:inDomain:` to search all domains available to your system. You can retrieve a list of potential browsing domains with NSNetServiceBrowser's `searchForRegistrationDomains` method by passing in the empty string (@" ") for the domain parameter. For more information read through "Browsing for Domains" (page 27)

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

To stop a search, use the `stop` method. You should perform any necessary cleanup in the `netServiceBrowserDidStopSearch:` delegate callback.

Listing 1 demonstrates how to browse for Bonjour network services with NSNetServiceBrowser. The code initializes the object, assigns a delegate, and begins a search for a hypothetical music service type, `_music._tcp`, on the local network. Also see the PictureSharingBrowser application in `/Developer/Examples/PictureSharingBrowser` for a good example of service browsing.

> **Note:** When the PictureSharingBrowser example code in Mac OS X v10.2 was finalized, the link-local suffix was `local.arpa.`, but it changed to `local.` before the operating system shipped. Use `local.` to specify the link-local network.

Listing 1 Browsing for Bonjour network services

```
id delegateObject; // Assume this exists.
NSNetServiceBrowser *serviceBrowser;

serviceBrowser = [[NSNetServiceBrowser alloc] init];
[serviceBrowser setDelegate:delegateObject];
[serviceBrowser searchForServicesOfType:@"_music._tcp" inDomain:@""];
```

Implementing Delegate Methods for Browsing

NSNetServiceBrowser returns all browsing results to its delegate. If you are using the class to browse for services, your delegate object should implement the following methods:

- `netServiceBrowserWillSearch:`
- `netServiceBrowserDidStopSearch:`
- `netServiceBrowser:didNotSearch:`
- `netServiceBrowser:didFindService:moreComing:`
- `netServiceBrowser:didRemoveService:moreComing:`

The `netServiceBrowserWillSearch:` method notifies the delegate that a search is commencing. You can use this method to update your user interface to reflect that a search is in progress. When browsing stops, the delegate receives a `netServiceBrowserDidStopSearch:` message, where you can perform any necessary cleanup.

If the delegate receives a `netServiceBrowser:didNotSearch:` message, it means that the search failed for some reason. You should extract the error information from the dictionary with the `NSNetServicesErrorCode` key and handle the error accordingly. See NSNetService for a list of possible errors.

You track services with the `netServiceBrowser:didFindService:moreComing:` and `netServiceBrowser:didRemoveService:moreComing:` methods, which indicate that a service has become available or has shut down. The “more coming” parameter indicates whether more results are on the way. If this parameter is `YES`, you should delay updating any user interface elements until the method is called with a “more coming” parameter of `NO`. However, if the parameter returns `NO` there is still a chance that more services will become available at a later time. It is important that you make sure to retain the

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

`didFindService` parameter before trying to resolve it, or you risk the object becoming deallocated. If you want a list of available services, you need to maintain your own array based on the information provided by delegate methods.

Listing 2 shows the interface for a class that responds to the NSNetServiceBrowser delegate methods required for service browsing, and Listing 3 shows its implementation. You can use this code as a starting point for your service browsing code.

Listing 2 Interface for an NSNetServiceBrowser delegate object (services)

```
#import <Foundation/Foundation.h>

@interface NetServiceBrowserDelegate : NSObject
{
    // Keeps track of available services
    NSMutableArray *services;

    // Keeps track of search status
    BOOL searching;
}

// NSNetServiceBrowser delegate methods for service browsing
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser;
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didNotSearch:(NSDictionary *)errorDict;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didFindService:(NSNetService *)aNetService
        moreComing:(BOOL)moreComing;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didRemoveService:(NSNetService *)aNetService
        moreComing:(BOOL)moreComing;

// Other methods
- (void)handleError:(NSNumber *)error;
- (void)updateUI;

@end
```

Listing 3 Implementation for an NSNetServiceBrowser delegate object (services)

```
#import "NetServiceBrowserDelegate.h"

@implementation NetServiceBrowserDelegate

- (id)init
{
    self = [super init];
    if (self) {
        services = [[NSMutableArray alloc] init];
        searching = NO;
    }
    return self;
}

- (void)dealloc
{
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
    [services release];
    [super dealloc];
}

// Sent when browsing begins
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser
{
    searching = YES;
    [self updateUI];
}

// Sent when browsing stops
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser
{
    searching = NO;
    [self updateUI];
}

// Sent if browsing fails
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didNotSearch:(NSDictionary *)errorDict
{
    searching = NO;
    [self handleError:[errorDict objectForKey:NSNetServicesErrorCode]];
}

// Sent when a service appears
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didFindService:(NSNetService *)aNetService
        moreComing:(BOOL)moreComing
{
    [services addObject:aNetService];
    if(!moreComing)
    {
        [self updateUI];
    }
}

// Sent when a service disappears
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didRemoveService:(NSNetService *)aNetService
        moreComing:(BOOL)moreComing
{
    [services removeObject:aNetService];

    if(!moreComing)
    {
        [self updateUI];
    }
}

// Error handling code
- (void)handleError:(NSNumber *)error
{
    NSLog(@"An error occurred. Error code = %d", [error intValue]);
    // Handle error here
}
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
// UI update code
- (void)updateUI
{
    if(searching)
    {
        // Update the user interface to indicate searching
        // Also update any UI that lists available services
    }
    else
    {
        // Update the user interface to indicate not searching
    }
}

@end
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Browsing for Domains

Bonjour service browsers and publishers that want to support hierarchical domain browsing should search for domains. If your application searches for services but does not care which domain it originates from, or, if your application publishes services and does not need to only publish to specific domains, you do not need to browse for domains. This article explains how to use the NSNetServiceBrowser class to discover available domains.

About Domain Browsing

The NSNetServiceBrowser class provides methods for browsing available domains. You should use these methods if you are writing a browser application that needs to see services outside the local network, or if you are publishing a service that needs to be seen outside the local network.

If you are writing a Bonjour network service browser, in general you should provide your users with the ability to browse all the domains available to them.

If the source domain of a service is not important, pass the empty string (@"") into the NSNetService and NSNetServiceBrowser methods that take a domain to find services. By passing the empty string for the domain, the system will automatically search the default domains as determined by the system's configuration and the network environment. In Mac OS X version 10.4, this means the local domain and possibly other global domains, whereas in Mac OS X version 10.2 and 10.3, this limits the search to the `local.` domain. Also, passing the empty string in domain searches will ensure that your application will be able to take advantage of future enhancements made to Bonjour. If you want to limit the domain which is searched in for services, just pass the name of the domain (such as `local.`) as the domain parameter.

Because domain browsing can take time, NSNetServiceBrowser objects perform browsing asynchronously by registering with a run loop. Browsing results are returned to your application through delegate methods. To correctly use an NSNetServiceBrowser object, you must assign it a delegate.

Browsing for domains takes three steps:

1. Initialize an NSNetServiceBrowser instance and assign a delegate to the object.
2. Begin a search for domains (either for registration or for browsing).
3. Handle search results and other messages sent to the delegate object.

The following sections describe these steps in detail.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Initializing the Browser and Starting a Search

To initialize an NSNetServiceBrowser object, use the `init` method. This sets up the browser and adds it to the current run loop. If you want to use a run loop other than the current one, use the `removeFromRunLoop:forMode:` and `scheduleInRunLoop:forMode:` methods.

If you want to search for domains in which you can register services, use the `searchForRegistrationDomains` method. If you want to find all domains available for browsing, use the `searchForBrowseDomains` method.

To stop a search, use the `stop` method. You should perform any necessary cleanup in the `netServiceBrowserDidStopSearch:` delegate callback.

Listing 1 demonstrates how to browse for registration domains with NSNetServiceBrowser. The code initializes the object, assigns a delegate, and begins a search for available domains.

Listing 1 Browsing for registration domains

```
id delegateObject; // Assume this exists.
NSNetServiceBrowser *domainBrowser;

domainBrowser = [[NSNetServiceBrowser alloc] init];
[domainBrowser setDelegate:delegateObject];
[domainBrowser searchForRegistrationDomains];
```

Implementing Delegate Methods for Browsing

NSNetServiceBrowser returns all browsing results to its delegate. If you are using the class to browse for domains, your delegate object should implement the following methods:

- `netServiceBrowserWillSearch:`
- `netServiceBrowserDidStopSearch:`
- `netServiceBrowser:didNotSearch:`
- `netServiceBrowser:didFindDomain:moreComing:`
- `netServiceBrowser:didRemoveDomain:moreComing:`

The `netServiceBrowserWillSearch:` method notifies the delegate that a search is commencing. You can use this method to update your user interface to reflect that a search is in progress. When browsing stops, the delegate receives a `netServiceBrowserDidStopSearch:` message, where you can perform any necessary cleanup.

If the delegate receives a `netServiceBrowser:didNotSearch:` message, it means that the search failed for some reason. You should extract the error information from the dictionary with the `NSNetServicesErrorCode` key and handle the error accordingly. See `NSNetServicesError` for a list of possible errors.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

You track domains with the `netServiceBrowser:didFindDomain:moreComing:` and `netServiceBrowser:didRemoveDomain:moreComing:` methods, which indicate that a service has become available or has shut down. The "more coming" parameter indicates whether more results are on the way. If this parameter is `YES`, you should not update any user interface elements until the method is called with a "more coming" parameter of `NO`. However, just because the parameter is `NO`, does not mean that more services will not become available in the future. If you want a list of available domains, you need to maintain your own array based on the information provided by delegate methods.

Listing 2 shows the interface for a class that responds to the NSNetServiceBrowser delegate methods required for domain browsing, and Listing 3 shows its implementation. You can use this code as a starting point for your domain browsing code.

Listing 2 Interface for an NSNetServiceBrowser delegate object (domains)

```
#import <Foundation/Foundation.h>

@interface NetServiceDomainBrowserDelegate : NSObject
{
    // Keeps track of available domains
    NSMutableArray *domains;

    // Keeps track of search status
    BOOL searching;
}

// NSNetServiceBrowser delegate methods for domain browsing
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser;
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didNotSearch:(NSDictionary *)errorDict;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didFindDomain:(NSString *)domainString
        moreComing:(BOOL)moreComing;
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didRemoveDomain:(NSString *)domainString
        moreComing:(BOOL)moreComing;

// Other methods
- (void)handleError:(NSNumber *)error;
- (void)updateUI;

@end
```

Listing 3 Implementation for an NSNetServiceBrowser delegate object (domains)

```
#import "NetServiceDomainBrowserDelegate.h"

@implementation NetServiceDomainBrowserDelegate

- (id)init
{
    self = [super init];
    if (self) {
        domains = [[NSMutableArray alloc] init];
        searching = NO;
    }
    return self;
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
}

- (void)dealloc
{
    [domains release];
    [super dealloc];
}

// Sent when browsing begins
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser
{
    searching = YES;
    [self updateUI];
}

// Sent when browsing stops
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser
{
    searching = NO;
    [self updateUI];
}

// Sent if browsing fails
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didNotSearch:(NSDictionary *)errorDict
{
    searching = NO;
    [self handleError:[errorDict objectForKey:NSNetServicesErrorCode]];
}

// Sent when a domain appears
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didFindDomain:(NSString *)domainString
        moreComing:(BOOL)moreComing
{
    [domains addObject:domainString];
    if(!moreComing)
    {
        [self updateUI];
    }
}

// Sent when a domain disappears
- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
        didRemoveDomain:(NSString *)domainString
        moreComing:(BOOL)moreComing
{
    [domains removeObject:domainString];

    if(!moreComing)
    {
        [self updateUI];
    }
}

// Error handling code
- (void)handleError:(NSNumber *)error
{
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
    NSLog(@"An error occurred. Error code = %@", error);
    // Handle error here
}

// UI update code
- (void)updateUI
{
    if(searching)
    {
        // Update the user interface to indicate searching
        // Also update any UI that lists available domains
    }
    else
    {
        // Update the user interface to indicate not searching
    }
}

@end
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Resolving and Using Network Services

This article describes how to use the NSNetService class to resolve discovered network services into socket information you can use to connect to a service.

The Resolution Process

The NSNetService class provides methods for resolving discovered services into network addresses and port numbers you can use to connect to a service. Resolution takes place every time a service is used because, although the service name is a persistent property, socket information (IP address and port number) can change from session to session. If a user browses for a service and saves that service, such as in a printer chooser, only the service name, type, and domain are stored. When it is time to connect, these values are resolved into socket information.

Because resolution can take time, especially if the service is unavailable, NSNetService resolves asynchronously, providing information to your application through a delegate object.

Resolving and using an NSNetService instance takes four steps:

1. Obtain a NSNetService instance through initialization or service discovery.
2. Resolve the service.
3. Respond to messages sent to the object's delegate about addresses or errors.
4. Use the resulting addresses to connect to the service.

The following sections describe these steps in detail.

Obtaining and Resolving an NSNetService Object

You can obtain an NSNetService object representing the service you want to connect to in one of two ways:

- Use NSNetServiceBrowser to discover services.
- Initialize a new NSNetService object with known name, type, and domain information, usually saved from a previous browsing session.

See "Browsing for Network Services" (page 21) for information about service browsing.

To create an NSNetService object for resolution rather than publication, use the `initWithDomain:type:name:` method.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Note: When browsing or resolving services, passing the empty string (@"") as the domain will specify the default list of domains, which includes local. and Back To My Mac. To limit your scope to a single domain or to access a domain not included in the default list, specify it explicitly, such as @"local.".

Once you have an NSNetService object to resolve, assign it a delegate and use the resolveWithTimeout: method to asynchronously search for socket addresses. When resolution is complete, the delegate receives a netServiceDidResolveAddress: or netService:didNotResolve: message if an error occurred. Because the delegate receives the identity of the NSNetService object as part of the delegate method, one delegate can serve multiple NSNetService objects.

Listing 1 demonstrates how to initialize and resolve an NSNetService object for a hypothetical music-sharing service. The code initializes the object with name serviceName, type _music._tcp, and the link-local suffix local.. It then assigns it a delegate and asks it to resolve the name into socket addresses.

Listing 1 Resolving network services with NSNetService

```
id delegateObject;      // Assume this exists.
NSString *serviceName;  // Assume this exists.
NSNetService *service;

service = [[NSNetService alloc] initWithDomain:@"local." type:@"_music._tcp"
                                name:serviceName];
[service setDelegate:delegateObject];
[service resolveWithTimeout:5.0];
```

Implementing Delegate Methods for Resolution

NSNetService returns resolution results to its delegate. If you are resolving a service, your delegate object should implement the following methods:

- netServiceDidResolveAddress:
- netService:didNotResolve:

The netServiceDidResolveAddress: method tells the delegate that the NSNetService object has added an address to its list of addresses for the service. However, more addresses may be added. For example, in systems that support both IPv4 and IPv6, netServiceDidResolveAddress: may be called two or more times: once for the IPv4 address and again for the IPv6 address. You should analyze the result of the addresses call before attempting to use the service to ensure that all the required connection information is present. If there's information missing, netServiceDidResolveAddress: will be called later. If multiple addresses are returned from resolving a service, try to connect to each one before giving up.

If resolution fails for any reason, the netService:didNotResolve: method is called. If the delegate receives a netService:didNotResolve: message, you should extract the type of error from the returned dictionary using the NSNetServicesErrorCode key and handle the error accordingly. See NSNetService for a list of possible errors.

Listing 2 shows the interface for a class that acts as a delegate for multiple NSNetService objects, and Listing 3 shows its implementation. You can use this code as a starting point for more sophisticated tracking of resolved services.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Listing 2 Interface for an NSNetService delegate object (resolution)

```
#import <Foundation/Foundation.h>

@interface NetServiceResolutionDelegate : NSObject
{
    // Keeps track of services handled by this delegate
    NSMutableArray *services;
}

// NSNetService delegate methods for publication
- (void)netServiceDidResolveAddress:(NSNetService *)netService;
- (void)netService:(NSNetService *)netService
        didNotResolve:(NSDictionary *)errorDict;

// Other methods
- (BOOL)addressesComplete:(NSArray *)addresses
        forServiceType:(NSString *)serviceType;
- (void)handleError:(NSNumber *)error withService:(NSNetService *)service;

@end
```

Listing 3 Implementation for an NSNetService delegate object (resolution)

```
#import "NetServiceResolutionDelegate.h"

@implementation NetServiceResolutionDelegate

- (id)init
{
    self = [super init];
    if (self) {
        services = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void)dealloc
{
    [services release];
    [super dealloc];
}

// Sent when addresses are resolved
- (void)netServiceDidResolveAddress:(NSNetService *)netService
{
    // Make sure [netService addresses] contains the
    // necessary connection information
    if ([self addressesComplete:[netService addresses]
            forServiceType:[netService type]]) {
        [services addObject:netService];
    }
}

// Sent if resolution fails
- (void)netService:(NSNetService *)netService
        didNotResolve:(NSDictionary *)errorDict
{
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
    [self handleError:[errorDict objectForKey:NSNetServicesErrorCode]];
    [services removeObject:netService];
}

// Verifies [netService addresses]
- (BOOL)addressesComplete:(NSArray *)addresses
        forServiceType:(NSString *)serviceType
{
    // Perform appropriate logic to ensure that [netService addresses]
    // contains the appropriate information to connect to the service
    return YES;
}

// Error handling code
- (void)handleError:(NSNumber *)error withService:(NSNetService *)service
{
    NSLog(@"An error occurred with service %@.%@.%@, error code = %@",
        [service name], [service type], [service domain], error);
    // Handle error here
}

@end
```

Connecting to a Network Service

Once an NSNetService object's delegate is notified that addresses have been resolved for the service, your application can connect. The recommended way to do this is with the getInputStream:outputStream: method of the NSNetService class. This method provides a reference to an input stream (NSInputStream) and an output stream (NSOutputStream), both of which you may access synchronously or asynchronously. To interact asynchronously, you need to schedule the streams in the current run loop and assign them a delegate object. These streams provide a protocol-independent means of communicating with network services.

Listing 4 demonstrates how to connect to an NSNetService using streams. Note that if you require only one of the two streams, it is not necessary for you to do anything with the other. See the PictureSharingBrowser application in /Developer/Examples/Foundation/PictureSharingBrowser for a complete example of service resolution with NSStream objects.

Listing 4 Connecting to a resolved Bonjour network service

```
#import <sys/socket.h>
#import <netinet/in.h>

// ...

NSNetService *service;  // Assume this exists. For instance, you may
                        // have received it from an NSNetServiceBrowser
                        // delegate callback.

NSInputStream *istream = nil;
NSOutputStream *ostream = nil;

[service getInputStream:&istream outputStream:&ostream];
if (istream && ostream)
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
{
    // Use the streams as you like for reading and writing.
}
else
{
    NSLog(@"Failed to acquire valid streams");
}
```

Monitoring a Service

If you want to monitor a service for changes, Mac OS X v10.4 has two new methods to provide this functionality. Monitoring a service could be useful in a situation such as a chat program. If a user (Matt) were to change his status from available to idle, the other users should learn this new information quickly, without having to constantly poll Matt's machine looking for his status. Enter `startMonitoring` and `stopMonitoring`.

By calling `startMonitoring` on your service Bonjour will monitor your service and call a delegate function if anything changes. The delegate method is named `netService:didUpdateTXTRecordData:` and passes the new TXT record as the didUpdateTXTRecordData parameter. When you no longer need to monitor the TXT record call `stopMonitoring` on your service.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

CFNetServices

CFNetServices is a Core Services API that allows you to register a network service, such as a printer or file server. By registering a service, 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.

The CFNetServices API uses the Core Services framework to provide access to Bonjour. It should be used when programming in C or C++ to write applications that need to discover services. If your C or C++ program uses a run loop, use this API.

Requirements

Version 10.2 or later of Mac OS X is required.

Using the CFNetServices API

The CFNetServices API provides access to Bonjour through three objects:

- `CFNetService` - An object that represents a single service on the network. A CFNetService has a name, a type, a domain, and a port number. Service types used by CFNetServices are maintained at http://www.dns-sd.org/servicetypes.html.
- `CFNetServiceBrowser` - An object used to discover domains and discover network services within domains.
- `CFNetServiceMonitor` - An object used to monitor services for changes to their TXT records.

Publishing a Service

Publishing a service on the network involves two tasks: creating a service, and registering a service. The next two sections will describe what is required to perform these two tasks.

Creating a CFNetService

The function that creates a CFNetService (`CFNetServiceCreate`) requires you to provide the following parameters that describe the service:

- Name — human-readable name of the service (such as "`Sales Laser Printer`")

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

- Service Type — the type of service, such as "`_printer._tcp`";
- Domain — the domain for the service, typically the empty string (`CFSTR("")`)for default domain(s), or `local.` for the local domain only
- Port — the port number the service listens on

Note: The dot in "`local.`" is part of the domain name. It signifies that the domain is fully qualified, which prevents anything from being added to the end of the domain (for example, `local.com`).

If you are implementing a protocol that relies on data stored in DNS text records, you can associate that information with a CFNetService by calling a separate CFNetServices function (`CFNetServiceSetTXTData`). Prior to Mac OS X v10.4, use the deprecated function `CFNetServiceSetProtocolSpecificInformation` instead.

Associate a callback function with your CFNetService by calling CFNetServices function `CFNetServiceSetClient`. Your callback function will be called to report errors that occur while your service is running.

If you want the service to run asynchronously, you must also schedule the CFNetService on a run loop by calling `CFNetServiceScheduleWithRunLoop`; otherwise, the service will run synchronously. For more information about the modes in which a service can run, see "Asynchronous and Synchronous Modes" (page 43).

Below is some example code for how to create a CFNetService.

Listing 1 Creating a CFNetService

```
CFNetService netService = CFNetServiceCreate(NULL, CFSTR(""), serviceType,
serviceName, chosenPort);
```

Registering a CFNetService

To make a service available on the network, call the CFNetServices function `CFNetServiceRegisterWithOptions` that registers a CFNetService. (This process is also known as "publishing" a service.) A CFNetServiceBrowser will be able to find the service until the service is stopped by the CFNetServices function `CFNetServiceCancel` that terminates services.

Please see Listing 2 (page 40) for sample code on this subject.

Listing 2 Registering an Asynchronous Service

```
void startBonjour (CFNetServiceRef netService) {
   CFStreamError error;
   CFNetServiceClientContext clientContext = { 0, NULL, NULL, NULL, NULL };

   CFNetServiceSetClient(netService, registerCallback, &clientContext);
   CFNetServiceScheduleWithRunLoop(netService, CFRunLoopGetCurrent(),
kCFRunLoopCommonModes);
   CFNetServiceRegister(netService, NULL);
   if (CFNetServiceRegister(netService, &error) == false) {
     CFNetServiceUnscheduleFromRunLoop(netService,
CFRunLoopGetCurrent(),kCFRunLoopCommonModes);
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
        CFNetServiceSetClient(netService, NULL, NULL);
        CFRelease(netService);
        fprintf(stderr, "could not register Bonjour service");
    }
}
```

Browsing for Services

To browse for services represented by a CFNetService, call the CFNetServices function that creates CFNetServiceBrowsers, `CFNetServiceBrowserCreate`. When you create a CFNetServiceBrowser, you need to provide a pointer to your callback function, which will be called when services are found.

If you want searches to be conducted asynchronously, you must also schedule the CFNetServiceBrowser on a run loop with`CFNetServiceBrowserScheduleWithRunLoop`.

To browse for services you can call the CFNetServices function `CFNetServiceBrowserSearchForServices` and specify the services to search for. For the domain parameter, you have two options. It is recommended that you pass the empty string (`CFSTR("")`) as the domain, allowing you to discover services in any domain on which your system is registered. Alternatively, you can specifying a domain to search in. Your callback function will be called and passed a CFNetService representing a matching service. The CFNetServiceBrowser will continue searching until your application stops the search by calling `CFNetServiceBrowserStopSearch`.

For each CFNetService that your callback function receives, you can call a CFNetServices function `CFNetServiceResolveWithTimeout` to update the CFNetService with the IP address for the service. Prior to Mac OS X v10.4, use the deprecated function `CFNetServiceResolve` instead of `CFNetServiceResolveWithTimeout`. Then call the CFNetService function `CFNetServiceGetAddressing` to get a CFArray containing a CFDataRef for each IP address associated with the service. Each CFDataRef consists of a `sockaddr` structure containing an IP address.

A good example of how to browse for services can be seen in Listing 3 (page 41).

Listing 3 Browsing Asynchronously for Services

```
static Boolean MyStartBrowsingForServices(CFStringRef type, CFStringRef domain)
 {
     CFNetServiceClientContext clientContext = { 0, NULL, NULL, NULL, NULL };
     CFStreamError error;
     Boolean result;

     assert(type != NULL);

     gServiceBrowserRef = CFNetServiceBrowserCreate(kCFAllocatorDefault,
MyBrowseCallBack, &clientContext);
     assert(gServiceBrowserRef != NULL);

     CFNetServiceBrowserScheduleWithRunLoop(gServiceBrowserRef,
CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

     result = CFNetServiceBrowserSearchForServices(gServiceBrowserRef, domain,
 type, &error);
     if (result == false) {

         // Something went wrong so lets clean up.
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
        CFNetServiceBrowserUnscheduleFromRunLoop(gServiceBrowserRef,
CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
CFRelease(gServiceBrowserRef);
        gServiceBrowserRef = NULL;

        fprintf(stderr, "CFNetServiceBrowserSearchForServices returned (domain
 = %d, error = %ld)\n", error.domain, error.error);
    }

    return result;
}
```

Resolving a Service

Once you have a name, type and domain, you can resolve the service to retrieve its host name and port. Like registering a service, resolving a service also needs to create a CFNetService reference by calling `CFNetServiceCreate`. Beginning in Mac OS X v10.4, When you call `CFNetServiceCreate` it is important to pass a valid domain that was obtained when the service was first detected. Starting with Mac OS X v10.4, if you pass an empty CFString for the domain argument, Bonjour will not equate it with the domain `local`. as it did in releases of Mac OS X prior to v10.4.

If you plan to resolve a service asynchronously, you should then associate the newly created `CFNetServiceRef` with a callback function, which will receive a `CFNetServiceRef`, and a pointer to a `CFStreamError`. The callback association is performed by calling `CFNetServiceSetClient` and following it with `CFNetServiceScheduleWithRunLoop` to add the service to a run loop. To use the current run loop, pass `CFRunLoopGetCurrent()` as a parameter.

After setting up the run loop, call the function `CFNetServiceResolve`, ensuring that it does not return an error. If an error is returned, then you should clean up all the references you created. Otherwise just wait for your callback functions to be called.

An example of resolving a service with CFNetService is in Listing 4 (page 42).

Listing 4 Resolving a Service Asynchronously

```
static void MyResolveService(CFStringRef name, CFStringRef type, CFStringRef
domain)
{
    CFNetServiceClientContext context = { 0, NULL, NULL, NULL, NULL };
    CFTimeInterval duration = 0; // use infinite timeout
    CFStreamError error;

    gServiceBeingResolved = CFNetServiceCreate(kCFAllocatorDefault, domain,
type, name, 0);
    assert(gServiceBeingResolved != NULL);

    CFNetServiceSetClient(gServiceBeingResolved, MyResolveCallback, &context);
   CFNetServiceScheduleWithRunLoop(gServiceBeingResolved, CFRunLoopGetCurrent(),
 kCFRunLoopCommonModes);

   if (CFNetServiceResolveWithTimeout(gServiceBeingResolved, duration, &error)
 == false) {

        // Something went wrong so lets clean up.
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

```
        CFNetServiceUnscheduleFromRunLoop(gServiceBeingResolved,
CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
        CFNetServiceSetClient(gServiceBeingResolved, NULL, NULL);
        CFRelease(gServiceBeingResolved);
        gServiceBeingResolved = NULL;

        fprintf(stderr, "CFNetServiceResolve returned (domain = %d, error =
%ld)\n", error.domain, error.error);
}

    return;
}
```

Monitoring a Service

`CFNetServiceMonitor`, which debuted in Mac OS X v10.4, gives developers the ability to watch services for changes to their TXT records. In order to monitor a service asynchronously, you need to follow the same general formula that is used for resolving a service.

Once you have a `CFNetServiceRef` for a service, you need to create a monitor reference (`CFNetServiceMonitorRef` using the function `CFNetServiceMonitorCreate`. Like resolving a service, the next step is to schedule the monitor reference on a run loop with `CFNetServiceMonitorScheduleWithRunLoop`. The default run loop can be obtained by calling `CFRunLoopGetCurrent()`.

After the monitor reference has been added to a run loop, you can start the monitor with the function `CFNetServiceMonitorStart` passing it the monitor reference. Make sure to check the return value to ensure that the monitor has started.

When you have finished monitoring a service, call `CFNetServiceMonitorStop` to stop the monitoring, `CFNetServiceMonitorUnscheduleFromRunLoop` to unschedule your monitor from its run loop and `CFNetServiceMonitorInvalidate` to destroy the monitor reference.

Asynchronous and Synchronous Modes

Several CFNetServices functions can operate in asynchronous or synchronous mode. Scheduling a CFNetService or a CFNetServiceBrowser on a run loop causes the service or browser to operate in asynchronous mode. If a CFNetService or a CFNetServiceBrowser is not scheduled on a run loop, it operates in synchronous mode. Operating in asynchronous mode changes the behavior of functions.

While it is possible to use the synchronous modes of these functions, please keep in mind that it is unacceptable to block the user interface or other functions of your program while you wait for synchronous functions to return. Due to the arbitrary amount of time network operations may last, it is highly recommended that you use the asynchronous modes of each function.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Table 1 Behavior of certain CFNetServices functions in asynchronous and synchronous mode

Function	Asynchronous mode	Synchronous mode
`CFNetServiceRegister-WithTimeout` (Mac OS X v10.4 only)	Starts the registration and returns. The callback function for the CFNetService will be called to report any errors that occur while the service is running. The service is available on the network until your application cancels the registration.	Blocks until your application cancels the service from another thread or until an error occurs, at which point the function returns. The error is returned in an error structure pointed to by a parameter of the `CFNetServiceRegisterWithOptions` function. The service is available on the network until your application cancels the registration or an error occurs.
`CFNetServiceResolve-WithTimeout` (Mac OS X v10.4 only)	Starts the resolution and returns. The callback function for the CFNetService will be called to report any errors that occur during resolution. The resolution process runs until the specified timeout is reached, or, if the timeout was specified as zero, until it is canceled.	Blocks until at least one IP address is found for the service, an error occurs, the time specified as the timeout parameter is reached or your application cancels the resolution, at which point the function returns. If an error occurs, the error is returned in an error structure pointed to by a parameter to the `CFNetServiceResolveWithTimeout` function. The resolution process continues to run until your application cancels it or an error occurs.
`CFNetServiceBrowser-SearchForDomains`	Starts the search and returns. The callback function for the CFNetServiceBrowser will be called for each domain that is found and to report any errors that occur while browsing. Browsing continues to run until your application stops the browsing.	Blocks until an error occurs or your application calls `CFNetServiceBrowser-StopSearch` at which time, the callback function for the CFNetServiceBrowser will be called for each domain that was found. Any error is returned in an error structure pointed to by a parameter to the `CFNetServiceBrowser-SearchForDomains` function. Browsing continues until your application stops the browsing.
`CFNetServiceBrowser-SearchForServices`	Starts the search and returns. The callback function for the CFNetServiceBrowser will be called for each CFNetService that is found and to report any errors that occur while browsing. Browsing continues to run until your application stops the browsing.	Blocks until an error occurs or until your application calls `CFNetServiceBrowser-StopSearch` at which time, the callback function for the CFNetServiceBrowser will be called for each CFNetService that was found. Any error is returned in an error structure pointed to by a parameter to the `CFNetServiceBrowser-SearchForServices` function. Browsing continues until your application stops the browsing.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Shutting Down Services and Searches

To shut down a service that is running in asynchronous mode, your application unschedules the service from all run loops it may be scheduled on and then calls `CFNetServiceSetClient` with the `clientCB` parameter set to `NULL` to disassociate your callback function from the CFNetService. Then call `CFNetServiceCancel` to stop the service. If the service is running in synchronous mode, you only need to call `CFNetServiceCancel` from another thread.

Listing 5 (page 45) shows a good example of how to shut down an asynchronous CFNetService Resolve process which has not timed out.

Listing 5 Canceling an Asynchronous CFNetService Resolve Process

```
void MyCancelResolve()
{
      assert(gServiceBeingResolved != NULL);
      CFNetServiceUnscheduleFromRunLoop(gServiceBeingResolved,
CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
      CFNetServiceSetClient(gServiceBeingResolved, NULL, NULL);
      CFNetServiceCancel(gServiceBeingResolved);
      CFRelease(gServiceBeingResolved);
      gServiceBeingResolved = NULL;
      return;
}
```

To shut down a browser that is running in asynchronous mode, your application unschedules the browser from all run loops it may be scheduled on and then calls `CFNetServiceBrowserInvalidate`. Then your application calls `CFNetServiceBrowserStopSearch`. If the browser is running in synchronous mode, you only need to call `CFNetServiceBrowserStopSearch`. An example of these functions can be seen in Listing 6 (page 45).

Listing 6 Stop Browsing for Services

```
static void MyStopBrowsingForServices()
{
      assert(gServiceBrowserRef != NULL);
      CFNetServiceBrowserStopSearch(gServiceBrowserRef, &streamerror);
      CFNetServiceBrowserUnscheduleFromRunLoop(gServiceBrowserRef,
CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
      CFNetServiceBrowserInvalidate(gServiceBrowserRef);
      CFRelease(gServiceBrowserRef);
      gServiceBrowserRef = NULL;
      return;
}
```


2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *NSNetServices and CFNetServices Programming Guide*.

Date	Notes
2010-03-24	Updated code examples to new initialization pattern.
2009-10-09	Corrected code listings.
2008-10-15	Updated code listings.
2005-11-09	Removed Preliminary stamp.
2005-10-04	New document that describes how to implement Bonjour in Cocoa or Carbon applications.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.

2010-03-24 | © 2010 Apple Inc. All Rights Reserved.