Discovering and Advertising Network Services

OS X and iOS provide four APIs for discovering and advertising network services:

- `NSNetService`—A high-level Objective-C API suitable for most app developers.
- `CFNetService`—A high-level C API suitable for use in Core Foundation code.
- DNS Service Discovery—A low-level C API suitable for cross-platform code. This API also offers more flexibility than the higher-level APIs.
- Game Kit framework—A high-level Objective-C API that provides peer-to-peer communication support for games, both locally (using infrastructure Wi-Fi and Bluetooth) and globally over the Internet.

In addition to these APIs, the Multipeer Connectivity Framework provides support for discovering and communicating with instances of your app and related apps on nearby devices using infrastructure Wi-Fi, peer-to-peer Wi-Fi, and either Bluetooth (for iOS) or Ethernet (for OS X).

As a rule, you should use Game Kit only for game-related peer-to-peer networking. For other peer-to-peer networking between iOS devices running iOS 7 and later, you should consider using the Multipeer Connectivity framework.

For compatibility with older versions of iOS, you can also write your own networking code and use `CFNetService` or `NSNetService` to advertise its availability.

Note: On devices that support Bluetooth, Bluetooth communication is automatically used by Game Kit. (Bonjour over Bluetooth can also be enabled when using the DNS Service Discovery C API by setting the `interfaceIndex` to `kDNSServiceFlagsIncludeP2P`. See *Bonjour over Bluetooth on iOS 5 and Later* for details.)

Bonjour Service Overview

A Bonjour service advertisement consists of three parts:

- Service name—This name must be unique to a particular instance of your program running on a particular computer.
- Service type—This must be the same for all instances of your program, and should be registered with the Internet Assigned Numbers Authority (IANA).
- Domain—If the domain value is empty, the host chooses the appropriate domains in which to publish or browse.

When an app browses for Bonjour services, it asks for services matching a particular type in a particular domain, and it gets back a list of matching service names. It should then present an appropriate UI to the user. When the user tells the app to connect to a particular service, the app should then connect to the service using a connect-to-service API. (If this is not possible for some reason, the app can pass the service's hostname and port to a connect-by-name API or, if a connect-by-name API is not available, the app can ask Bonjour to resolve the hostname, and the app can then connect by IP address and port number.)

Publishing a Network Service

Bonjour zero-configuration networking lets you advertise network services, such as a printer or a document syncing service, on a network. There are three ways to publish a network service:

- For Objective-C and Core Foundation code, the recommended way is with the `CFNetServices` API.

- For portable C code that must run on operating systems other than iOS and OS X, the DNS Service Discovery C API is recommended.

You can publish a network service with the following steps:

1. Create a socket to listen for connections to the service. See Writing a TCP-Based Server in *Networking Programming Topics* for the recommended way to listen for connections on a network socket.
2. Create a service object, providing the port of your socket, the domain (usually an empty string), and the service type string of your choosing:
   - With Foundation, initialize an `NSNetService` object with the `initWithDomain:type:name:port:` method.
   - With Core Foundation, create a `CFNetServiceRef` object with the `CFNetServiceCreate` function.
   - With the DNS Service Discovery API, call `DNSServiceRegister` to return a `DNSServiceRef` object.
3. Assign a delegate or callback:
   - With Foundation, assign a delegate to the `NSNetService` object with the `delegate` method.
   - With Core Foundation, assign a client callback to the `CFNetServiceRef` object with the `CFNetServiceSetClient` function.
   - With the DNS Service Discovery API, you should pass a client callback (and, optionally, a pointer to a context object of your choosing) in your call to `DNSServiceRegister`. At this point, you are done except for handling callbacks when they occur.
4. Schedule or reschedule the service, if necessary:
   - With Foundation, the service is automatically scheduled on the current run loop in the default mode. If you need to schedule the object on another run loop or in a different mode, you should unschedule it and reschedule it at this point.
   - With Core Foundation, you *must* schedule the `CFNetServicesRef` object on a run loop by calling `CFNetServiceScheduleWithRunLoop`.
   - With the DNS Service Discovery API, call `DNSServiceSetDispatchQueue` to schedule the service on a dispatch queue. (If you must support an OS prior to OS X v10.7, see the *SRVResolver* sample code project for an example of how to use DNS Service Discovery without Grand Central Dispatch.)
5. Publish the service, if necessary:
   - With Foundation, publish the service by calling the `publish` method.
   - With Core Foundation, publish the service by calling `CFNetServiceRegisterWithOptions`.
   - With the DNS Service Discovery API, no further action is necessary; the service was already published when you called `DNSServiceRegister`.

After your service is published, you can listen for connections on your socket and set up input and output streams when a connection is made.

Important: If you create a custom protocol, you should use a custom service type, and register that service type with IANA. For details, see RFC 6335.

Browsing for and Connecting to a Network Service

The process for finding and resolving a network service is as simple as the process for publishing one. To browse for network services in Objective-C, create an instance of the `NSNetServiceBrowser` class and assign it a delegate. Then, call the `searchForServicesOfType:inDomain:` method on the service browser. The `netServiceBrowser:didFindService:moreComing:` delegate method is called once for every service found.

To connect to a service, first stop the browsing by calling `stop` (unless you have a specific reason to keep browsing), then call the `getInputStream:outputStream:` method on the `NSNetService` object that represents the service. The address of the service is resolved automatically.

You can also use the `CFStreamCreatePairWithSocketToNetService` function with a `CFNetServiceRef` object to connect to a Bonjour service.

> **Important:** If you are using ARC, you should read *NSNetService and Automatic Reference Counting (ARC)*.

Resolving a Network Service

You may need to resolve a network service manually to provide the service's address to an API that does not accept network service names. To resolve a network service in Objective-C, first stop the browsing by calling `stop` (unless you have a specific reason to keep browsing), then call the `resolveWithTimeout:` method on the `NSNetService` object that represents the service.

The `netServiceDidResolveAddress:` method is called on the service's delegate when the service's address has been resolved. You can then access the service's hostname with the `hostName` method or its address information with the `addresses` method. To avoid unnecessary network traffic, you should also call `stop` on the `NSNetService` object as soon as it returns a set of addresses.

> **Important:** The resolution process returns both numerical IP addresses and a hostname. The IP addresses can be an arbitrary mix of IPv4 and IPv6 addresses. Unless you are doing something unusual, you should normally pass the hostname to any API that supports hostnames rather than using the IP addresses directly, because otherwise you would otherwise have to write your own code to try connecting to each of the multiple IP addresses in parallel or in series (described further in Avoid Resolving DNS Names Before Connecting to a Host).
>
> The resolver caches the mapping from hostname to IP addresses, so future lookups do not result in any additional network traffic.

Multipeer Connectivity Overview

The Multipeer Connectivity Framework provides a layer on top of Bonjour that lets you communicate with apps running on nearby devices (over infrastructure Wi-Fi, peer-to-peer Wi-Fi, and either Bluetooth (for iOS) or Ethernet (for OS X) without having to write lots of networking code specific to your app.

With Multipeer Connectivity, your app advertises its availability. It can then discover other instances of your app (or other apps that share the same service type) running on nearby devices, and can invite those nearby peers to join a session. If they accept the invitation, your app can send messages and files to one or more of the connected peers with just a single method call.

> **Important:** As with Bonjour, your app must provide a service type, and you should register that service type with IANA. For details, see RFC 6335.

If you need stream-based communication, your app can open a unidirectional stream to any connected peer (which can also open a unidirectional stream back to your app in response).

Finally, Multipeer Connectivity provides the ability to share small amounts of data (such as the user's screen name) outside the context of a session, if desired, allowing you to provide the user with information that he or she can use when choosing peers to invite into a session.

To Learn More

Multipeer Connectivity—Read *Multipeer Connectivity Framework Reference* and the *MultipeerGroupChat* sample code project.

Game Kit—Read *Game Center Programming Guide*, *Game Kit Framework Reference*, and the *GKRocket* and *GKTank* sample code projects.

`NSNetService`—Read *NSNetServices and CFNetServices Programming Guide*, *NSNetServiceBrowser Class Reference*, *NSNetServiceBrowserDelegate Protocol Reference*, and *NSNetServiceDelegate Protocol Reference*. For sample code, see the *RemoteCurrency* sample code project.

`CFNetService`—Read *NSNetServices and CFNetServices Programming Guide* and *CFNetServices Reference*.

DNS Service Discovery—Read *DNS Service Discovery Programming Guide* and *DNS Service Discovery C Reference*.