Getting Started with Bonjour

This thread has been locked by a moderator.

292

Every now and then I talk to someone who's trying to use Bonjour and just can't get over the first hurdle. That happened today, and so I decided to share my write-up for the benefit of others.

Questions or comments? Put them in a new thread here on DevForums, tagging it with *Bonjour* so that I see it.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Getting Started with Bonjour

Bonjour is an Apple term for a variety of Internet standards [1]. Bonjour allows your app to browse for and connect to services on the network without infrastructure support. For example, Bonjour lets you find and connect to a printer even if the network has no DHCP server to hand out IP addresses.

If you're new to Bonjour, a good place to start is the Bonjour Overview. It's in the documentation archive, so it hasn't been updated in a while, but the fundamentals haven't changed.

There are, however, two things that *have* changed:

- Network framework has new Bonjour APIs, and the old ones are now deprecated.
- iOS 14 introduced local network privacy.

This post shows how to get started with Bonjour, taking into account these new developments.

[1] Specifically:

- RFC 3927 Dynamic Configuration of IPv4 Link-Local Addresses
- RFC 6762 Multicast DNS
- RFC 6763 DNS-Based Service Discovery

Start Browsing

Let's start by implementing a service browser. To simplify things, this browses for SSH services. That way you can get started with the browser without first having to implement a server to register your service. If you don't already have an SSH service registered on your network, start one by enabling System Settings > General > Sharing > Remote Login on your Mac.

The SSH service type is, unsurprisingly, `_ssh._tcp`. First, on your Mac, run the `dns-sd` tool to confirm that you have an SSH service visible on your network:

```
% dns-sd -B "_ssh._tcp" "local."
% dns-sd -B "_ssh._tcp" "local."
…
Timestamp     A/R    Flags  if Domain  Service Type  Instance Name
…
11:54:43.315  Add        2   6 local.  _ssh._tcp.    Fluffy
…
11:54:43.725  Add        2   6 local.  _ssh._tcp.    SAM the Robot 12
^C
```

This shows that I have two services, one called *Fluffy* and the other called *SAM the Robot 12*. Let's write some iOS code to browse for those. To start, create an app from the iOS > App template and connect a button to the `startStop()` method of a class like this:

```
import Foundation
import Network

class AppModel {

    var browserQ: NWBrowser? = nil

    func start() -> NWBrowser {
        print("browser will start")
        let descriptor = NWBrowser.Descriptor.bonjour(type: "_ssh._tcp", domain: "local.")
        let browser = NWBrowser(for: descriptor, using: .tcp)
        browser.stateUpdateHandler = { newState in
            print("browser did change state, new: \(newState)")
        }
        browser.browseResultsChangedHandler = { updated, changes in
            print("browser results did change:")
            for change in changes {
                switch change {
                case .added(let result):
                    print("+ \(result.endpoint)")
                case .removed(let result):
                    print("- \(result.endpoint)")
                case .changed(old: let old, new: let new, flags: _):
                    print("± \(old.endpoint) \(new.endpoint)")
                case .identical:
                    fallthrough
                @unknown default:
                    print("?")
                }
            }
        }
        browser.start(queue: .main)
        return browser
    }

    func stop(browser: NWBrowser) {
        print("browser will stop")
        browser.stateUpdateHandler = nil
        browser.cancel()
    }

    func startStop() {
        if let browser = self.browserQ {
            self.browserQ = nil
            self.stop(browser: browser)
        } else {
            self.browserQ = self.start()
        }
    }
}
```

Note I'm using SwiftUI, but if you chose to use UIKit you could add this code directly to your view controller. Of course, whether you *want* to add networking code to your view controller is another question. The answer is, natch, "No", except when creating a tiny test project like this one (-:

Now build and run **in the simulator** and click your buton. It'll print something like this:

```
browser will start
browser did change state, new: ready
browser results did change:
+ SAM the Robot 12._ssh._tcp.local.
+ Fluffy._ssh._tcp.local.
```

As you can see, it's found our two SSH services. Yay!

Run on the Device

Now stop the app and run it on a real device. This time the Test button results in:

```
browser will start
…
browser did change state, new: failed(-65555: NoAuth)
```

This is local network privacy kicking in. There are two things you need to do:

- Add a `NSBonjourServices` property to your `Info.plist` to declare what service types you're using.
- Add a `NSLocalNetworkUsageDescription` property to your `Info.plist` to explain what you're doing with the local network.

Do that and run your app again. On tapping the Test button you'll see an alert asking you to grant your app access to the local network. Tap Allow and the browser will start generating results as before.

Respond to Updates

When working with Bonjour it's important to keep your browser running to update your app's state. To test this, start a Remote Login on a different machine and look for a new result being printed:

```
browser results did change:
+ Slimey._ssh._tcp.local.
```

And then turn it off:

```
browser results did change:
- Slimey._ssh._tcp.local.
```

If you don't have another Mac to test this with, start a dummy service using `dns-sd`:

```
% dns-sd -R "Guy Smiley" "_ssh._tcp" "local." 12345
Registering Service Test._ssh._tcp.local. port 12345
…
```

Press control-C to stop the `dns-sd` tool, which unregisters the service.

Connect

When the user choose a service, it's time to connect. There are two ways to do this, depending on the networking API you use to run your connection.

`NWConnection` can connect directly to a Bonjour service endpoint. For example, you might have code that connects to a DNS name and port:

```
func makeConnection(host: String, port: UInt16) -> NWConnection {
    let host = NWEndpoint.Host(host)
    let port = NWEndpoint.Port(rawValue: port)!
    let endpoint = NWEndpoint.hostPort(host: host, port: port)
    return NWConnection(to: endpoint, using: .tcp)
}
```

Replace that with code that takes the endpoint you get back from the browser:

```
func makeConnection(endpoint: NWEndpoint) -> NWConnection {
    return NWConnection(to: endpoint, using: .tcp)
}
```

If you're using a legacy API, like BSD Sockets, you'll need to resolve the Bonjour service endpoint to a DNS name and then pass that DNS name into your connection code. Network framework does not support resolving Bonjour service endpoints out of the box, so you'll have to do that yourself. For an example of how you might do this, see this post.

IMPORTANT For this to work reliably, your BSD Sockets code must support Happy Eyeballs. See TN3151 Choosing the right networking API for specific advice on that front.

Register a Service

Now let's look at the server side. To listen for connections with Network framework, you might write code like this:

```
import Foundation
import Network

class AppModel {

    var listenerQ: NWListener? = nil

    func start() -> NWListener? {
        print("listener will start")
        guard let listener = try? NWListener(using: .tcp) else { return nil }
        listener.stateUpdateHandler = { newState in
            print("listener did change state, new: \(newState)")
        }
        listener.newConnectionHandler = { connection in
            connection.cancel()
        }
        listener.start(queue: .main)
        return listener
    }

    func stop(listener: NWListener) {
        print("listener will stop")
        listener.stateUpdateHandler = nil
        listener.cancel()
    }

    func startStop() {
        if let listener = self.listenerQ {
            self.listenerQ = nil
            self.stop(listener: listener)
        } else {
            self.listenerQ = self.start()
        }
    }
}
```

To register your service with Bonjour, add these lines before the call to `start(queue:)`:

```
        listener.service = .init(type: "_ssh._tcp")
        listener.serviceRegistrationUpdateHandler = { change in
            print(change)
        }
```

The listener calls your service registration update handler to tell you the name of the service. Typically you display this value somewhere in your UI. For more about this, see Showing Connection Information in an iOS Server.

To confirm that your service is running, open Terminal and choose Shell > New Remote Command. Your service should show up in the Secure Shell (ssh) list.

Alternatively, browse for SSH services using the `dns-sd` tool, as illustrated in the *Start Browsing* section above.

Network Bonjour

Reply

Posted 2 months ago by eskimo

Add a Comment