Showing Connection Information in an iOS Server

This thread has been locked by a moderator.

For important background information, read Extra-ordinary Networking before reading this.

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

Showing Connection Information in an iOS Server

A common pattern for iOS apps is to implement an embedded HTTP server in the app that allows the user to interact with it over the network. For example, a comic book reader app might have a web server that let’s the user upload comics to the app using Safari.

The networking aspect of this is fairly straightforward. You don’t need to do anything extra-ordinary with network interfaces and so on. Rather, start an ordinary listener and register it with Bonjour. In Network framework that involves setting the `service` property on the listener, and there are equivalent techniques in other APIs.

Where things get complicated is in the UI; or, more accurately, in determining the information to display in the UI. An app like this typically wants to display three bits of information to the user:

- Bonjour service name, like *Guy Smiley* — If the user has a Bonjour-aware browser [1], this is by far the easiest way to connect.
- Local DNS name, like `guy-smiley.local.` — This works in virtually all browsers on all platforms, and is relatively easy for the user to enter.
- IP addresses — Typing IP addresses is very old school, but sometimes that’s the user’s only option.

The following sections explain how to get each of these items.

[1] Sadly, that doesn’t include Safari.

Bonjour Service Name

All Bonjour APIs have a mechanism to tell you about the name that they used to register your service. For example, with Network framework, the listener has a `serviceRegistrationUpdateHandler` property. The listener calls that closure with status information about the registration, including the service name.

IMPORTANT Bonjour automatically renames your service if it discovers a name conflict. Write your code to handle that case, updating your UI if the service name changes.

Local DNS Name

Getting the local DNS name on iOS is tricky. The best option is to run a Bonjour service resolution query against your own service name. You can’t use standard service resolution APIs, like `DNSServiceResolve`, because those stop when the resolution completes, and you want to continue monitoring for changes. Rather, use the `DNSServiceQueryRecord` function to start an ongoing query for the service’s SRV record. Restrict this query to the local interface (`kDNSServiceInterfaceIndexLocalOnly`) to avoid generating unnecessary traffic on the ‘wire’.

Note macOS makes this much easier because you can use System Configuration framework to get the local DNS name. `SCDynamicStoreCopyLocalHostName` returns the current value. To learn about changes, use the dynamic store’s notification mechanism to watch for changes to the `Setup:/System` key.

IP Addresses

As discussed in Network Interface Concepts, there is no single IP address you can display here. Rather, you have to be prepared to display multiple addresses.

To get this list of IP addresses:

1. Get a list of all IP addresses, grouped by interface.
2. Filter that list for interfaces where the functional type is either Wi-Fi or wired.
3. Filter it again for interfaces that have at least one IPv4 address.
4. Merge those lists.

This list may include IPv6 addresses. Those are hard to display and even harder for the user to type. Consider whether or not to show them in your UI.

Note What about IPv6-only interfaces, I hear you ask? It turns out that for Wi-Fi and wired interfaces that the user cares about, the system will assign a link-local IPv4 address to the interface. So, step 3 is effectively filtering for interfaces that the user cares about. Neat, eh?

For information about the specific APIs to use here, see Network Interface APIs.

IMPORTANT Update your UI when the IP address list changes. Use `kNotifySCNetworkChange` to drive this update. See Network Interface APIs for the details.

Note This approach only makes sense for iOS and its child platforms. On macOS, use System Configuration framework for this:

- Use an `SCNetworkInterface` object to get a list of network interfaces and their various properties.
- Use an `SCDynamicStore` object to get the IP address list and monitor for changes.

Network

Reply

Posted 2 days ago by eskimo

Add a Comment