

Don't Try to Get the Device's IP Address

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"

54

Don't Try to Get the Device's IP Address

I regularly see questions like:

- How do I find the IP address of the device?
- How do I find the IP address of the Wi-Fi interface?
- How do I identify the Wi-Fi interface?

I also see a lot of *really* bad answers to these questions. That's understandable, because the questions themselves don't make sense. Networking on Apple platforms is complicated and many of the things that are 'obviously' true are, in fact, not true at all. For example:

- There's no single IP address that represents the device, or an interface. A device can have 0 or more interfaces, each of which can have 0 or more IP addresses, each of which can be IPv4 and IPv6.
- A device can have multiple interfaces of a given type. It's common for iPhones to have multiple WWAN interfaces, for example.

It's not possible to give a simple answer to any of these questions, because the correct answer depends on the context. Why do you need this particular information? What are you planning to do with it?

This post describes the scenarios I most commonly encounter, with my advice on how to handle each scenario.

IMPORTANT BSD interface names, like `en0`, are not considered API. There's no guarantee, for example, that an iPhone's Wi-Fi interface is `en0`. If you write code that relies on a hard-coded interface name, it will fail in some situations.

Service Discovery

Some folks want to identify the Wi-Fi interface so that they can run a custom service discovery protocol over it. Before you do that, I strongly recommend that you look at Bonjour. This has a bunch of advantages:

- It's an industry standard [1].
- It's going to be more efficient on the 'wire'.
- You don't have to implement it yourself, you can just call an API [2].

For information about the APIs available, see TN3151 [Choosing the right networking API](#).

If you must implement your own service discovery protocol, don't think in terms of finding *the* Wi-Fi interface. Rather, write your code to work with *all* Wi-Fi interfaces, or perhaps even all Ethernet-like interfaces. That's what Apple's Bonjour implementation does, and it means that things will work in odd situations [3].

To find all Wi-Fi interfaces, get the interface list and filter it for ones with the Wi-Fi functional type. To find all broadcast capable interfaces, get the interface list and filter it for interfaces with the `IFF_BROADCAST` flag set. If the service you're trying to discover only supports IPv4, filter out any IPv6-only interfaces.

For advice on how to do this, see *Interface List* and *Network Interface Type* in [Network Interface APIs](#).

When working with multiple interfaces, it's generally a good idea to create a socket per interface and then bind that socket to the interface. That ensures that, when you send a packet, it'll definitely go out the interface you expect.

[1] Bonjour is an Apple term for:

- RFC 3927 [Dynamic Configuration of IPv4 Link-Local Addresses](#)
- RFC 6762 [Multicast DNS](#)
- RFC 6763 [DNS-Based Service Discovery](#)

[2] That's true even on non-Apple platforms. It's even true on most embedded platforms. If you're talking to a Wi-Fi accessory, see [Working with a Wi-Fi Accessory](#).

[3] Even if the service you're trying to discover can only be found on Wi-Fi, it's possible for a user to have their iPhone on an Ethernet that's bridged to a Wi-Fi. Why on earth would they do that? Well, security, of course. Some organisations forbid their employees from using Wi-Fi.

Logging and Diagnostics

Some folks want to log the IP address of the Wi-Fi interface, or the WWAN, or both for diagnostic purposes. This is quite feasible, with the only caveat being there may be multiple interfaces of each type.

To find all interfaces of a particular type, get the interface list and filter it for interfaces with that functional type. See *Interface List* and *Network Interface Type* in [Network Interface APIs](#).

Interface for an Outgoing Connection

There are situations where you need to get the interface used by a particular connection. A classic example of that is FTP. When you set up a transfer in FTP, you start with a control connection to the FTP server. You then open a listener and send its IP address and port to the FTP server over your control connection. What IP address should you use?

There's an easy answer here: Use the local IP address for the control connection. That's the one that the server is most likely to be able to connect to.

To get the local address of a connection:

- In Network framework, first get the `currentPath` [property](#) and then get its `localEndpoint` [property](#).
- In BSD Sockets, use `getsockname`. See its [man page](#) for details.

Now, this isn't a particularly realistic example. Most folks don't use FTP these days [1] but, even if they do, they use FTP passive mode, which avoids the need for this technique. However, this sort of thing still does come up in practice. I recently encountered two different variants of the same problem:

- One developer was implementing VoIP software and needed to pass the devices IP address to their VoIP stack. The best IP address to use was the local IP address of their control connection to the VoIP server.
- A different developer was upgrading the firmware of an accessory. They do this by starting a server within their app and sending a command to the accessory to download the firmware from that server. Again, the best IP address to use is the local address of the control connection.

[1] See the discussion in TN3151 [Choosing the right networking API](#).

Listening for Connections

If you're listening for incoming network connections, you don't need to bind to a specific address. Rather, listen on all local addresses. In Network framework, this is the default for `NWListener`. In BSD Sockets, set the address to `INADDR_ANY` (IPv4) or `in6addr_any` (IPv6).

If you only want to listen on a specific interface, don't try to bind to that interface's IP address. If you do that, things will go wrong if the interface's IP address changes. Rather, bind to the interface itself:

- In Network framework, set either the `requiredInterfaceType` [property](#) or the `requiredInterface` [property](#) on the `NWParameters` you use to create your `NWListener`.
- In BSD Sockets, set the `IP_BOUND_IF` (IPv4) or `IPV6_BOUND_IF` (IPv6) socket option.

How do you work out what interface to use? The standard technique is to get the interface list and filter it for interfaces with the desired functional type. See *Interface List* and *Network Interface Type* in [Network Interface APIs](#). Remember that their may be multiple interfaces of a given type. If you're using BSD Sockets, where you can only bind to a single interface, you'll need to create multiple listeners, one for each interface.

Listener UI

Some apps have an embedded network server and they want to populate a UI with information on how to connect to that server. This is a surprisingly tricky task to do correctly. For the details, see [Showing Connection Information for a Local Server](#).

Outgoing Connections

In some situations you might want to force an outgoing connection to run over a specific interface. There are four common cases here:

- Set the local address of a connection [1].
- Force a connection to run over a specific interface.
- Force a connection to run over a type of interface.
- Force a connection to run over an interface with specific characteristics. For example, you want to download some large resource without exhausting the user's cellular data allowance.

The last case should be the most common — see the *Constraints* section of [Network Interface Techniques](#) — but all four are useful in specific circumstances.

The following sections explain how to tackle these tasks in the most common networking APIs.

[1] This implicitly forces the connection to use the interface with that address. For an explanation as to why, see the discussion of scoped routing in [Network Interface Techniques](#).

Network Framework

Network framework has good support for all of these cases. Set one or more of the following properties on the `NWParameters` object you use to create your `NWConnection`:

- `requiredLocalEndpoint` [property](#)
- `requiredInterface` [property](#)
- `prohibitedInterfaces` [property](#)
- `requiredInterfaceType` [property](#)
- `prohibitedInterfaceTypes` [property](#)
- `prohibitConstrainedPaths` [property](#)
- `prohibitExpensivePaths` [property](#)

Foundation URL Loading System

`URLSession` has fewer options than Network framework but they work in a similar way: Set one or more of the following properties on the `URLSessionConfiguration` object you use to create your session:

- `allowsCellularAccess` [property](#)
- `allowsConstrainedNetworkAccess` [property](#)
- `allowsExpensiveNetworkAccess` [property](#)

Note While these session configuration properties are also available on `URLRequest`, it's better to configure this on the session.

There's no option that forces a connection to run over a specific interface. In most cases you don't need this — it's better to use the `allowsConstrainedNetworkAccess` and `allowsExpensiveNetworkAccess` properties — but there are some situations where that's necessary. For advice on this front, see [Running an HTTP Request over WWAN](#).

BSD Sockets

BSD Sockets has very few options in this space. One thing that's easy and obvious is setting the local address of a connection: Do that by passing the address to `bind`.

Alternatively, to force a connection to run over a specific interface, set the `IP_BOUND_IF` (IPv4) or `IPV6_BOUND_IF` (IPv6) socket options.

Network

Reply

Posted 2 days ago by eskimo

Add a Comment

This site contains user submitted content, comments and opinions and is for informational purposes only. Apple disclaims any and all liability for the acts, omissions and conduct of any third parties in connection with or related to your use of the site. All postings and use of the content on this site are subject to the [Apple Developer Forums Participation Agreement](#).

Platforms	Topics & Technologies	Resources	Programs
IOS	Accessibility	Documentation	Apple Developer Program
iPadOS	Accessories	Curriculum	Apple Developer Enterprise Program
macOS	App Extensions	Downloads	App Store Small Business Program
tvOS	App Store	Forums	MFi Program
watchOS	Audio & Video	Videos	News Partner Program
	Augmented Reality		Video Partner Program
	Business		Security Bounty Program
Swift	Design	Support	Security Research Device Program
SwiftUI	Distribution	Support Articles	
SF Symbols	Education	Contact Us	
Swift Playgrounds	Fonts	Bug Reporting	
TestFlight	Games	System Status	
Xcode	Health & Fitness		
Xcode Cloud	In-App Purchase	Account	App Accelerators
	Localization	Apple Developer	App Store Awards
	Maps & Location	App Store Connect	Apple Design Awards
	Machine Learning	Certificates, IDs, & Profiles	Apple Developer Academies
	Security	Feedback Assistant	Entrepreneur Camp
	Safari & Web		Tech Talks
			WWDC