Network Interface APIs

This thread has been locked by a moderator.

30

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"`

Network Interface APIs

Most developers don’t need to interact directly with network interfaces. If you do, read this post for a summary of the APIs available to you.

Before you read this, read Network Interface Concepts.

Interface List

The standard way to get a list of interfaces and their addresses is `getifaddrs`. To learn more about this API, see its man page.

A network interface has four fundamental attributes:

- A set of flags — These are packed into a `CUnsignedInt`. The flags bits are declared in `<net/if.h>`, starting with `IFF_UP`.
- An interface type — See *Network Interface Type*, below.
- An interface index — Valid indexes are greater than 0.
- A BSD interface name. For example, an Ethernet interface might be called `en0`. The interface name is shared between multiple network interfaces running over a given hardware interface. For example, IPv4 and IPv6 running over that Ethernet interface will both have the name `en0`.

WARNING BSD interface names are not considered API. There’s no guarantee, for example, that an iPhone’s Wi-Fi interface is `en0`.

You can map between the last two using `if_indextoname` and `if_nametoindex`. See the `if_indextoname` man page for details.

An interface may also have address information. If present, this always includes the interface address (`ifa_addr`) and the network mask (`ifa_netmask`). In addition:

- Broadcast-capable interfaces (`IFF_BROADCAST`) have a broadcast address (`ifa_broadaddr`, which is an alias for `ifa_dstaddr`).
- Point-to-point interfaces (`IFF_POINTOPOINT`) have a destination address (`ifa_dstaddr`).

Calling `getifaddrs` from Swift is a bit tricky. For an example of this, see QSocket: Interfaces.

IP Address List

Once you have `getifaddrs` working, it’s relatively easy to manipulate the results to build a list of just IP addresses, a list of IP addresses for each interface, and so on. QSocket: Interfaces has some Swift snippets that show this.

Interface List Updates

The interface list can change over time. Hardware interfaces can be added and removed, network interfaces come up and go down, and their addresses can change. It’s best to avoid caching information from `getifaddrs`. If thats unavoidable, use the `kNotifySCNetworkChange` Darwin notification to update your cache. For information about registering for Darwin notifications, see the `notify` man page (in section 3).

This notification just tells you that *something* has changed. It’s up to you to fetch the new interface list and adjust your cache accordingly.

You’ll find that this notification is sometimes posted numerous times in rapid succession. To avoid unnecessary thrashing, debounce it.

While the Darwin notification API is easy to call from Swift, Swift does not import `kNotifySCNetworkChange`. To fix that, define that value yourself, calling a C function to get the value:

```
var kNotifySCNetworkChange: UnsafePointer<CChar> {
    networkChangeNotifyKey()
}
```

Here’s what that C function looks like:

```
extern const char * networkChangeNotifyKey(void) {
    return kNotifySCNetworkChange;
}
```

Network Interface Type

There are two ways to think about a network interface’s type. Historically there were a wide variety of weird and wonderful types of network interfaces. The following code gets this legacy value for a specific BSD interface name:

```
func legacyTypeForInterfaceNamed(_ name: String) -> UInt8? {
    var addrList: UnsafeMutablePointer<ifaddrs>? = nil
    let err = getifaddrs(&addrList)
    // In theory we could check `errno` here but, honestly, what are gonna
    // do with that info?
    guard
        err >= 0,
        let first = addrList
    else { return nil }
    defer { freeifaddrs(addrList) }
    return sequence(first: first, next: { $0.pointee.ifa_next })
        .compactMap { addr in
            guard
                let nameC = addr.pointee.ifa_name,
                name == String(cString: nameC),
                let sa = addr.pointee.ifa_addr,
                sa.pointee.sa_family == AF_LINK,
                let data = addr.pointee.ifa_data
            else { return nil }
            return data.assumingMemoryBound(to: if_data.self).pointee.ifi_type
        }
        .first
}
```

The values are defined in `<net/if_types.h>`, starting with `IFT_OTHER`.

However, this value is rarely useful because many interfaces ‘look like’ Ethernet and thus have a type of `IFT_ETHER`.

Network framework has the concept of an interface’s *functional type*. This is an indication of how the interface fits into the system. There are two ways to get an interface’s functional type:

- If you’re using Network framework and have an `NWInterface` value, get the `type` property.
- If not, call `ioctl` with a `SIOCGIFFUNCTIONALTYPE` request. The return values are defined in `<net/if.h>`, starting with `IFRTYPE_FUNCTIONAL_UNKNOWN`.

Swift does not import `SIOCGIFFUNCTIONALTYPE`, so it’s best to write this code in a C:

```
extern uint32_t functionalTypeForInterfaceNamed(const char * name) {
    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd < 0) { return IFRTYPE_FUNCTIONAL_UNKNOWN; }

    struct ifreq ifr = {};
    strlcpy(ifr.ifr_name, name, sizeof(ifr.ifr_name));

    bool success = ioctl(fd, SIOCGIFFUNCTIONALTYPE, &ifr) >= 0;

    int junk = close(fd);
    assert(junk == 0);

    if ( ! success ) { return IFRTYPE_FUNCTIONAL_UNKNOWN; }

    return ifr.ifr_ifru.ifru_functional_type;
}
```

Network

Reply

Posted 2 days ago by eskimo

Add a Comment