Chapter 35. Geolocation without GPS

I made things easy for myself in the last chapter by relying on a GPS receiver to generate latitude and longitude information. In this chapter, I throw away the GPS device (figuratively, of course), and determine my position and geographical address (my *geolocation*) without its help.

Although this seems a daunting task, I'm encouraged by the availability of geolocation in HTML5. Even when there's no GPS receiver attached to the computer, a tiny piece of JavaScript executed by the browser can return your latitude and longitude. A mapping example using HTML5's geolocation API can be found at http://html5demos.com/geo. The Google map it displays for my position is shown in Figure 1.

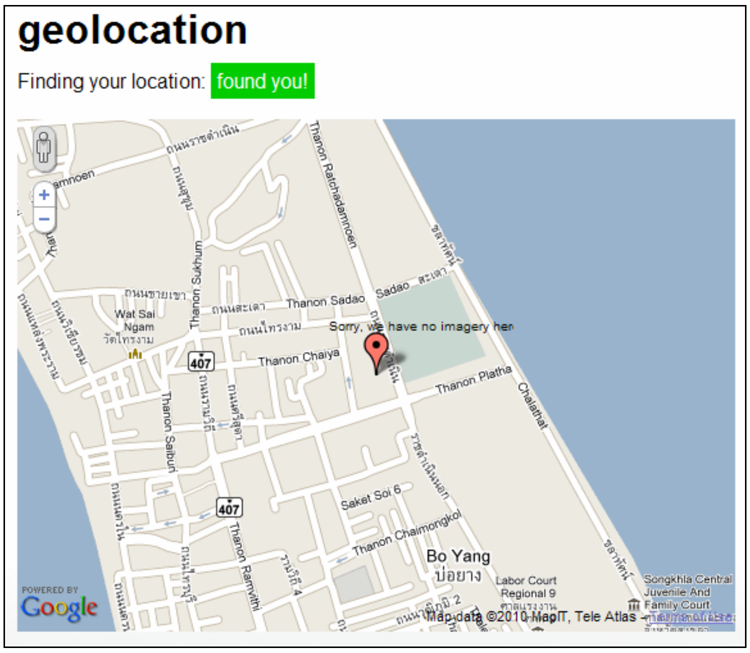


Figure 1. An HTML5-generated Map.

This map isn't such an impressive display of HTML5's magic, since it places me in the wrong city – Songkhla, about 25 km east of my true location, at Prince of Songkla University (PSU) in Hat Yai. To be fair, US and European users have reported more impressive results, and the API makes no guarantees about returning an accurate location.

How does HTML5 work? At the programming level, the Web page author employs the navigator.geolocation JavaScript property, and a callback function that's executed after the user agrees to share information over the Web. A good introduction to the coding details can be found in Chapter 6 of the online book "Dive into HTML5" by Mark Pilgrim (http://diveintohtml5.org/geolocation.html).

The W3C Geolocation API Specification (http://www.w3.org/TR/geolocation-API), states that an implementation can use several alternative location sources, depending on the capabilities of the device. Possibilities include GPS, triangulation via cellular tower IDs, RFID and Bluetooth IDs, nearby WiFi access points, and the computer’s IP address. Firefox utilizes this information by sending it to Google's Location Services which returns a location estimate (http://www.mozilla.com/en-US/firefox/geolocation/).

I want to emulate HTML5's geolocation API in Java without having to fire-up a browser. Also, I'm going to restrict my hardware to a netbook without a GPS receiver, RFID or Bluetooth; the machine's only connection to the wider world will be through WiFi. This leaves two starting points for finding my location: IP and MAC addresses.

What are IP and MAC Addresses?

As you probably know, an IP address is a unique ID assigned to any device connected to a TCP/IP network (such as the Internet). By assuming that computers are fairly immobile, databases have been compiled that map IP addresses to locations. Unfortunately, an increasing number of machines aren't fixed in one place, and/or utilize non-unique, dynamic IP addresses.

A Media Access Control (MAC) address is meant to be a permanent, unique ID assigned to a network interface card (NIC) or LAN card on your machine. But MAC addresses aren't as permanent as we might hope; often the address is stored in a card's firmware, which makes it possible to modify (called *MAC spoofing*). The address doesn't contain a latitude or longitude, but details on the card's manufacturer and model. The IEEE maintains a database of vendors, called the Organizationally Unique Identifiers (OUI), which can be searched using the first three bytes of a MAC address at http://standards.ieee.org/develop/regauth/oui/public.html.

How can a MAC address be converted into a latitude and longitude? For that we must thank Google and Skyhook Wireless (http://www.skyhookwireless.com/). They've been busy driving around cities and towns in North America, Europe, Asia, and Australia, plotting the location of WiFi access points (APs) and cell towers. Skyhook has a map online showing its coverage (http://www.skyhookwireless.com/howitworks/coverage.php), and claims their database has over 250 million entries.

This occupation is often called *wardriving*, and has been getting Google into trouble for 'snooping'. It's probably okay to record a wireless' name (SSID) and MAC address, along with an associated latitude and longitude. The problems begin if data being sent over those networks is recorded.

Although IP and MAC addresses are quite different, TCP/IP includes protocols for converting from IP to MAC addresses -- the Address Resolution Protocol (ARP) for IPv4 and the Neighbor Discovery Protocol (NDP) for IPv6.

Using IP and MAC addresses

Figure 2 shows an overview of the various ways I'll be using IP and MAC addresses in this chapter.

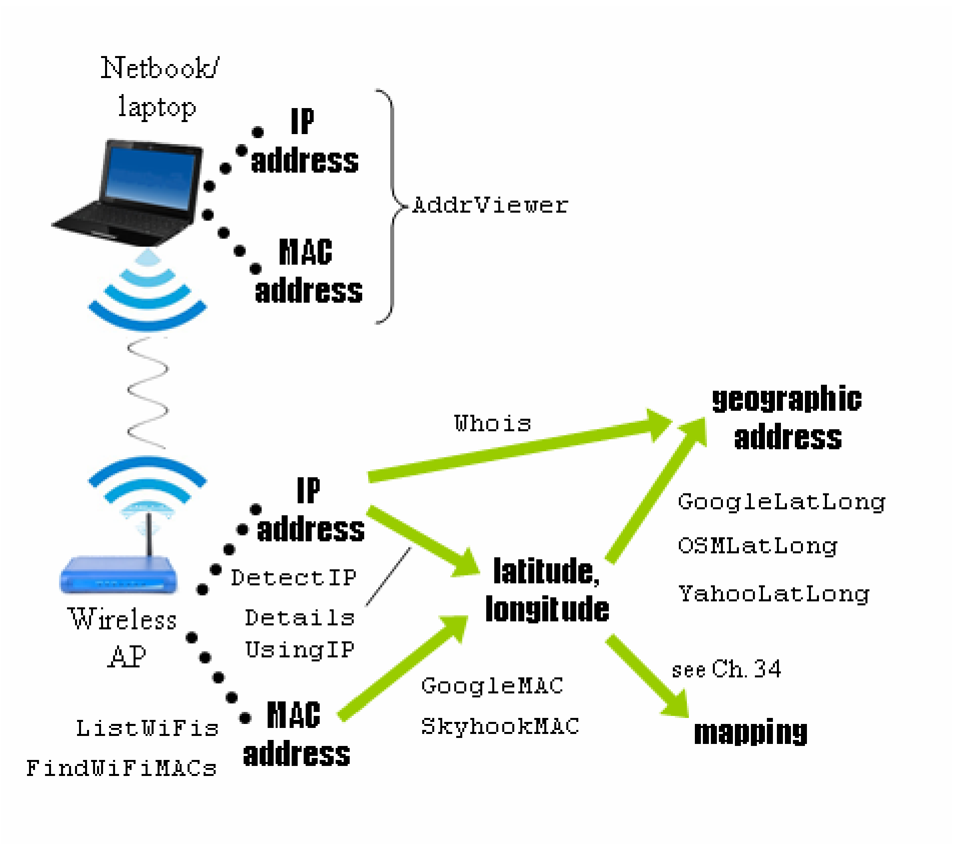


Figure 2. From IP/MAC addresses to Locations.

The IP and MAC addresses of my netbook aren't much use, because they're for a mobile device. Instead I have to reach out across the wireless network for the addresses of WiFi Access Points (APs). These APs are probably in fixed positions, and so likely to be listed in location databases.

The dotted lines and arrows in Figure 2 have Java program names next to them. For example, I'll describe two programs, GoogleMAC.java and SkyhookMAC.java, for converting a MAC address into a latitude and longitude. As their names suggest, one uses Google, the other Skyhook Wireless.

I'll explore several approaches since they tend to give slightly different answers, with the quality depending on the AP's location and the coverage of the databases involved. For example, SkyhookMAC.java fails to return any information for my netbook, probably because of Skyhook's sketchy coverage of the south of Thailand.

Most of the programs are similar – they utilize Web services, and parse the JSON, XML, or HTML responses. For that reason, it's a good idea if you read Chapter 33 on Web Service APIs first (http://fivedots.coe.psu.ac.th/~ad/jg/ch33/), because I won't be explaining those details again. I also won't be describing latitude and longitude mapping, because I just talked about that in Chapter 34 (http://fivedots.coe.psu.ac.th/~ad/jg/ch34/).

1. Examining my Netbook

Two JavaSE network classes, InetAddress and NetworkInterface, provide all I need to list my netbook's IP and MAC addresses, although NetworkInterface. getHardwareAddress() for reading the MAC address was only added in JavaSE 6. More detailed information can be displayed for each NIC via the InterfaceAddress class (also new to JavaSE 6). A good example of its capabilities can be found at http://stackoverflow.com/questions/494465/how-to-enumerate-ip-addresses-of-all-enabled-nic-cards-from-java

The following code comes from my AddrViewer.java. showIP() prints the machine's IP address:

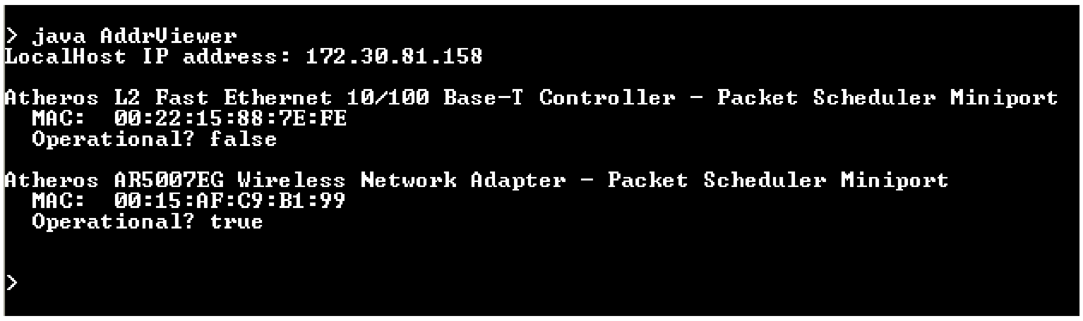
```
private static void showIP()
{
  try {
    InetAddress localHost = InetAddress.getLocalHost();
    System.out.println("LocalHost IP address: " +
                     localHost.getCanonicalHostName() + "\n");
  }
  catch (UnknownHostException e)
  {  System.out.println(" No LocalHost Info found\n"); }

}  // end of showIP()
```

showMacNICs() iterates through the NICs on a device, printing their MAC addresses:

```
private static void showMacNICs()
// list the NICs that have MAC addresses
{
  try {
    Enumeration<NetworkInterface> intfs =
                     NetworkInterface.getNetworkInterfaces();
    while(intfs.hasMoreElements()) {
      NetworkInterface intf = intfs.nextElement();
      byte[] macAddr = intf.getHardwareAddress();
      if (macAddr != null) {
        System.out.println(intf.getDisplayName());
        System.out.println("  MAC:  " + macToString(macAddr));
        System.out.println("  Operational? " + intf.isUp());
        System.out.println();
      }
    }
  }
  catch (SocketException e)
  {  e.printStackTrace(); }
}  // end of showMacNICs()
```

NetworkInterface.getDisplayName() supplies the NIC's SSID (its name). NetworkInterface.getHardwareAddress() unhelpfully returns a MAC address as a byte array, so macToString() converts it to a string containing two-digit hexadecimals separated by ":"s.

Information for my netbook is shown in Figure 3.



```
> java AddrViewer
LocalHost IP address: 172.30.81.158

Atheros L2 Fast Ethernet 10/100 Base-T Controller - Packet Scheduler Miniport
  MAC:  00:22:15:88:7E:FE
  Operational? false

Atheros AR5007EG Wireless Network Adapter - Packet Scheduler Miniport
  MAC:  00:15:AF:C9:B1:99
  Operational? true


>
```

Figure 3. IP and MAC addresses for my Netbook.

The netbook has an Ethernet port (currently not in use) and a WiFi card whose MAC address is 00:15:AF:C9:B1:99. If the first three hexadecimal digits are used to query the IEEE OUI database (http://standards.ieee.org/develop/regauth/oui/public.html), the vendor is revealed to be AzureWave Technologies from Taiwan.

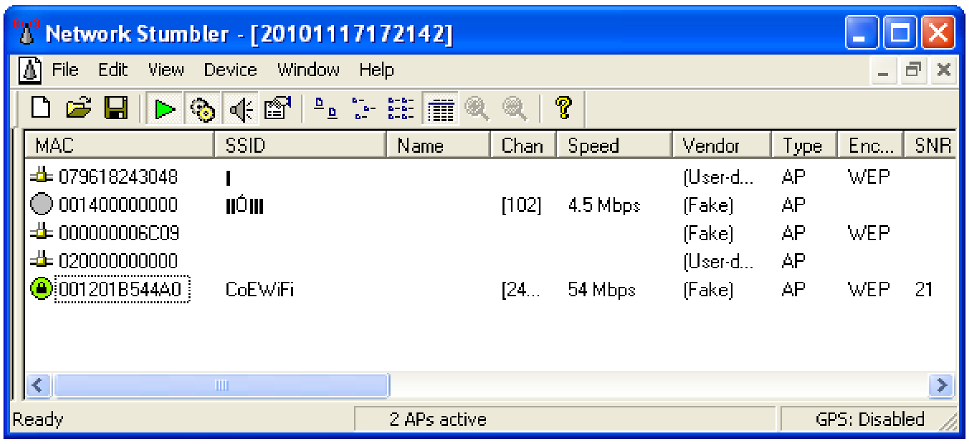
Supplying the MAC address to GoogleMAC.java (which I'll explain later) produces a latitude and longitude of (6.867817, 101.249477) which is in Pattani in the south of Thailand, about 90 km from my true location. The netbook's IP address, 172.30.81.158, was dynamically assigned when I connected to the ISP, so is of no use.

If you don't want to write Java code such as AddrViewer.java for examining your device, then a handy existing tool on most platforms is ipconfig (ipconfig /all on Windows, ipconfig –a on Linux).

2. MAC Hack Attack on Wireless Access Points

It's time for me to don my Hacker gear – black wraparound sunglasses and hoodie – and to dowse my office lights, apart from a cool red spot shining on my keyboard. Yo, it's time for wardriving!

To be truthful, wardriving isn't that difficult. The easiest GUI tool for the job is NetStumbler (http://www.netstumbler.com/), which produces details shown in Figure 4 on my netbook.



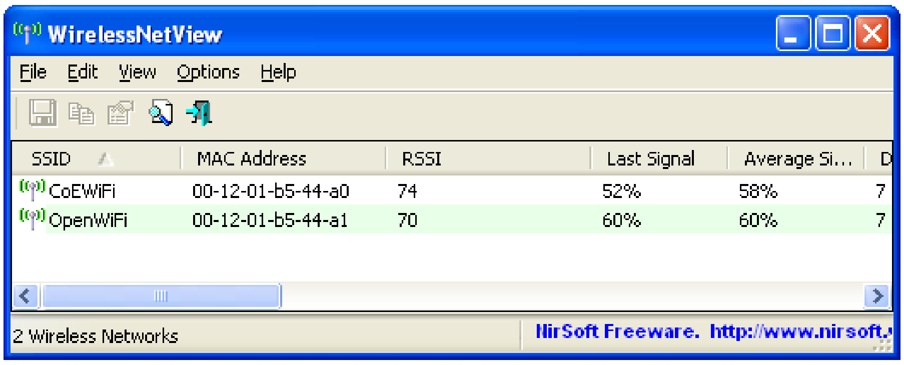
MAC	SSID	Name	Chan	Speed	Vendor	Type	Enc...	SNR
079618243048	I				(User-d...	AP	WEP	
001400000000	IIÓIII		[102]	4.5 Mbps	(Fake)	AP		
000000006C09					(Fake)	AP	WEP	
020000000000					(User-d...	AP		
001201B544A0	CoEWiFi		[24...	54 Mbps	(Fake)	AP	WEP	21

Ready 2 APs active GPS: Disabled

Figure 4. NetStumbler on my Netbook.

NetStumbler is an “active” wireless network detector, which sends out Probe Request messages (frames), and receives Probe Responses from APs. Another approach is “passive” detection, which doesn't broadcast signals. Instead, the programs (e.g. Kismet, AirSnort) listen for 802.11 traffic within range of the wireless card.

WirelessNetView (http://www.nirsoft.net/utils/wireless_network_view.html) is a simpler "active" tool than NetStumbler, and has a command line interface that will be useful later. Each detected WiFi network is listed with its SSID, MAC Address, RSSI (received signal strength), and other useful details. Figure 5 shows it running on my netbook.



SSID	MAC Address	RSSI	Last Signal	Average Si...	D
CoEWiFi	00-12-01-b5-44-a0	74	52%	58%	7
OpenWiFi	00-12-01-b5-44-a1	70	60%	60%	7

2 Wireless Networks NirSoft Freeware. http://www.nirsoft.

Figure 5. WirelessNetView on my Netbook.

In Figure 5, it detected the university's "OpenWifi" AP which NetStumbler didn't list, but missed some of the department's private WiFis. Both tools saw "CoEWiFi", the WiFi I'm using.

3. Collecting WiFi Details with Java

At this point, I could content myself with NetStumbler or WirelessNetView, and use pen and paper to make notes of the MAC addresses they find. Instead, I'll explain two Java solutions which collect these addresses.

3.1. The Place Lab API

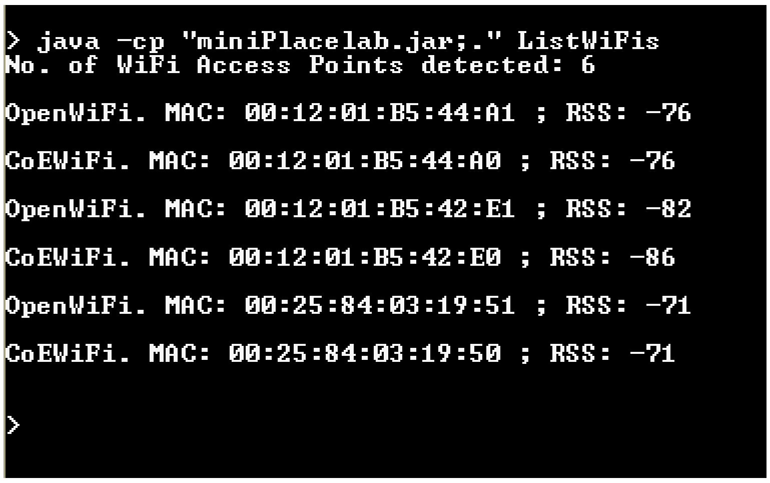
Place Lab (http://www.placelab.org/) is a location API that works in a similar way to HTML5's geolocation API. It utilizes a range of techniques, such as WiFi APs, GSM cell phone towers, and fixed Bluetooth devices to obtain MAC addresses. The derived positions can be rendered on a locally cached map.

The Windows download (placelab-win32-2.1.zip) is 14.42 MB large, but much of this isn't needed for obtaining an Access Point's MAC address. A Place Lab NDIS network protocol driver must be installed, and then only placelab.jar is required in order to implement a "WiFi spotter". Placelab.jar is nearly 5 MB in size, but it's possible to delete many of the classes and packages inside that JAR which deal with mapping, databases, IDE integration, JavaME, and others. The resulting JAR is 196 KB, and could easily be trimmed further.

The Place Lab download comes with a nice collection of examples, including WiFiSpotterExample.java (http://www.placelab.org/toolkit/doc/#wifispotterexample), which became the basis of my ListWiFis.java application:

```
public static void main(String[] args)
{
  Spotter spotter = new WiFiSpotter();
  try {
    spotter.open();
    BeaconMeasurement bm =
       (BeaconMeasurement) spotter.getMeasurement();
    int numAPs = bm.numberOfReadings();
    System.out.println("No. of WiFi Access Points detected: " +
                                      numAPs + "\n");
    if (numAPs > 0) {
      WiFiReading wr;  // iterate through the readings
      for (int i=0; i < numAPs; i++) {
        wr = (WiFiReading) bm.getReading(i);
        String macAddr = wr.getId().toUpperCase();
                  // use uppercase hex
        System.out.println(wr.getSsid() + ". MAC: " +  macAddr +
                            " ; RSS: " + wr.getRssi() + "\n");
      }
    }
  }
  catch (SpotterException e)
  {  e.printStackTrace();  }
}  // end of main()
```

ListWiFis.java utilizes the Place Lab WiFiSpotter class, which can access all the 802.11 wireless cards on the machine and collect details about the connected networks. A reading is obtained with BeaconMeasurement.getReading(), and cast to a WiFiReading so each AP’s SSID, MAC address, and RSS can be printed. Figure 6 shows typical output.



```
> java -cp "miniPlacelab.jar;." ListWiFis
No. of WiFi Access Points detected: 6

OpenWiFi. MAC: 00:12:01:B5:44:A1 ; RSS: -76

CoEWiFi. MAC: 00:12:01:B5:44:A0 ; RSS: -76

OpenWiFi. MAC: 00:12:01:B5:42:E1 ; RSS: -82

CoEWiFi. MAC: 00:12:01:B5:42:E0 ; RSS: -86

OpenWiFi. MAC: 00:25:84:03:19:51 ; RSS: -71

CoEWiFi. MAC: 00:25:84:03:19:50 ; RSS: -71


>
```

Figure 6. Listing WiFi Details with Place Lab.

The simplest way of understanding negative RSS values is to remember that a smaller negative number is better (i.e. is a stronger signal), and a typical strength range is -70 to -90 (-70 is better). The negatives are due to the use of a logarithmic scale, where negative numbers denote small, positive strengths. For example, with a $\log_{10}$ scale, a value of -2 represents 10^{-2}, which equals 0.01; by the same reasoning -4 denotes 0.0001.

3.2. Utilizing WirelessNetView from Java

A somewhat hacky alternative way of obtaining MAC addresses is to have Java execute the WirelessNetView tool (http://www.nirsoft.net/utils/wireless_network_view.html), and read its results. WirelessNetView has a simple command line interface which Java can easily utilize.

I began by writing a DOS batch file, called applyWNV.bat, to execute WirelessNetView with a file argument for storing its sorted output.

```
@echo off

If [%1]==[] goto Error

echo Storing WiFi AP info in %1 by using WirelsssNetView...
WirelessNetView /scomma %1 /sort "~Average Signal"

echo Finished.
exit /b

:Error
echo Usage: applyWNV ^<out_file^>
exit /b
```

This batch file approach allowed me to test the script separately from Java. In addition, I can employ variables and control flow operations (e.g. if) which are trickier from the command line.

A typical call to the batch file would be:

```
> applyWNV.bat wifiInfo.txt
```

The wifiInfo.txt output file will contain multiple lines, one for each AP detected. For example:

```
CoEWiFi,62%,62%,1,Yes,Yes,RSNA,CCMP,ERP,
11/22/2010 11:58:46 AM,11/22/2010 11:58:46 AM,
00-25-84-03-19-50,-71,2.437,6,,54 Mbps
```

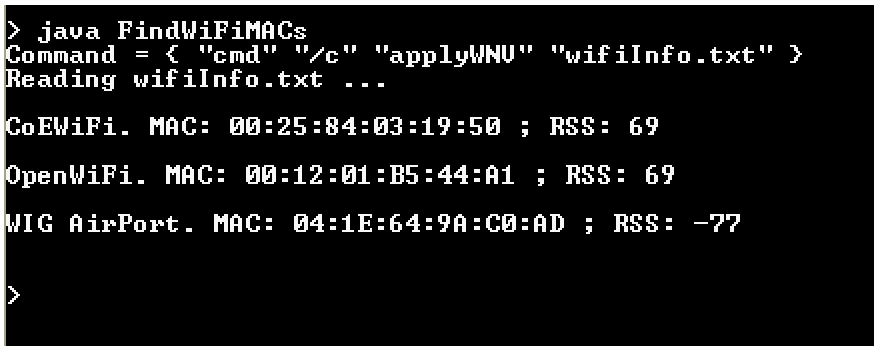
The arguments are separated by commas due to "/scomma" argument in the call to WirelessNetView. I'm interested in each AP's SSID, MAC address, and RSS, which are the first, 12th, and 13th values on a line (shown in bold above).

Calling applyWNV.bat from Java is possible with the java.lang.ProcessBuilder class, but the coding can be tricky depending on the type of OS command that you're trying to invoke. Instead, I used my SaferExec class, which hides many ProcessBuilder problems behind a simpler interface. A description of SaferExec can be found at http://fivedots.coe.psu.ac.th/~ad/SaferExec/.

FindWiFiMACs.java uses SaferExec to call the applyWNV.bat script with a file name. The file's resulting contents are read in, and each AP's SSIDs, MAC address and RSS are printed. The code for FindWiFiMACs.java is:

```
// global
private static final String INFO_FNM = "wifiInfo.txt";


public static void main (String[] args)
{
  SaferExec se = new SaferExec();
  se.exec("cmd /c applyWNV " + INFO_FNM);
              // cmd /c necessary for calling a batch file

  try {
    System.out.println("Reading " + INFO_FNM + " ...");
    System.out.println();
    BufferedReader in = new BufferedReader(new FileReader(INFO_FNM));
    String wifiLine;
    while ((wifiLine = in.readLine()) != null)
      extractMACDetails(wifiLine);
    in.close();
  }
  catch (IOException e)
  { System.out.println("Problem reading " + INFO_FNM);  }
 }  // end of main


private static void extractMACDetails(String wifiLine)
{
  String[] args = wifiLine.split(",");
  if (args.length != 17)
    System.out.println("Incorrect number of arguments");
  else {
    String macAddr = args[11].toUpperCase().replaceAll("-", ":");
                // use uppercase hex and a ":" as separator
    System.out.println(args[0] + ". MAC: " +  macAddr +
                              " ; RSS: " + args[12] + "\n");
  }
}  // end of extractMACDetails()
```

extractMACDetails() pulls apart each input line by calling String.split() with ",". The MAC address is slightly reformatted before being printed out, as shown in Figure 7.



```
> java FindWiFiMACs
Command = < "cmd" "/c" "applyWNV" "wifiInfo.txt" >
Reading wifiInfo.txt ...

CoEWiFi. MAC: 00:25:84:03:19:50 ; RSS: 69

OpenWiFi. MAC: 00:12:01:B5:44:A1 ; RSS: 69

WIG AirPort. MAC: 04:1E:64:9A:C0:AD ; RSS: -77


>
```

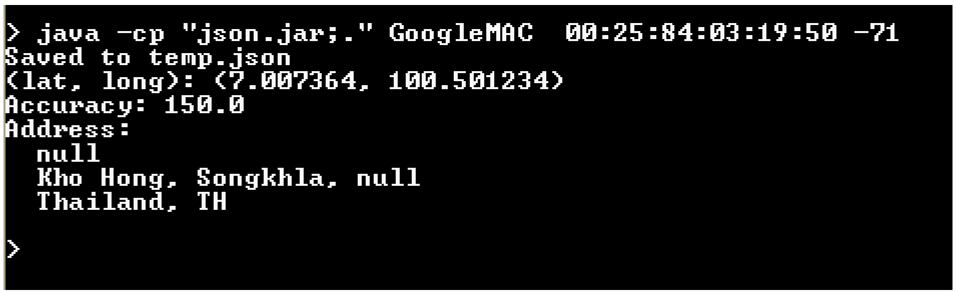
Figure 7. Listing WiFi Details with WirelessNetView and Java..

4. From MAC Addresses to Locations

Armed with MAC addresses, I can start searching Google's and Skyhook Wireless' MAC-to-location databases.

4.1. Googling for MAC Addresses

GoogleMac.java posts a query to Google's Location Services REST API (http://www.google.com/loc/json), supplying a MAC address and an optional signal strength. It returns JSON data containing latitude, longitude, and a geographical address. Figure 8 shows the program's output when supplied with the last "CoEWiFi" MAC address from Figure 6.



```
> java -cp "json.jar;." GoogleMAC  00:25:84:03:19:50 -71
Saved to temp.json
(lat, long): (7.007364, 100.501234)
Accuracy: 150.0
Address:
  null
  Kho Hong, Songkhla, null
  Thailand, TH

>
```

Figure 8. Using Google to Lookup a MAC Address.

The generated address details are non too impressive since city information is missing, although the district (Kho Hong) is correct. However, the latitude and longitude are within 100 m from where I was sitting when I obtained the MAC address. Figure 9 shows a Google map view of (7.007364, 100.501234).

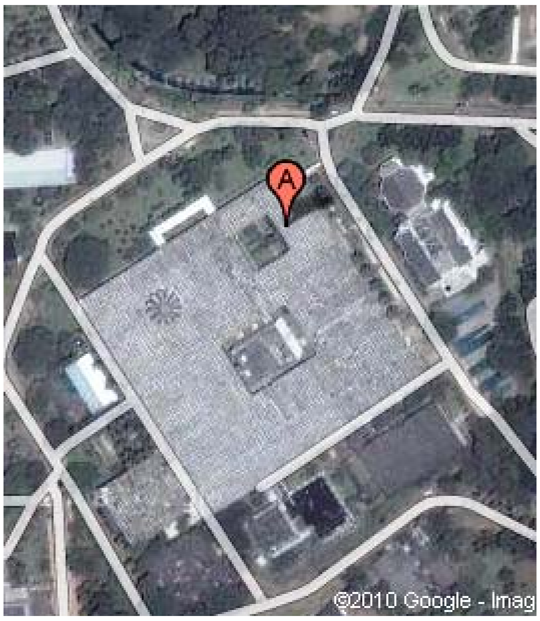


Figure 9. Google Map of (7.007364, 100.501234).

The large square building in Figure 9 is the Faculty of Engineering at PSU.

GoogleMAC.java posts the user's supplied MAC address and signal strength to http://www.google.com/loc/json formatted as follows:

```
{"version":"1.1.0",
 "request_address":true,
 "wifi_towers":[{"mac_address":"00:25:84:03:19:50",
                 "ssid":"", "signal_strength":-71} ]}
```

The "request_address" parameter signals that address details should be returned along with the latitude and longitude. Multiple tuples can be included in the "wifi_towers" list, which may let Google triangulate a position based on the signal strengths of the APs. The "ssid" field can be left out if it isn't assigned a value.

This format is described in more detail in the blog posts by E-rant at http://www.e-rant.net/2010/06/google-wi-fi-hullabalooo/ and Tin Isles at http://tinisles.blogspot.com/2009/12/wifi-geolocation.html.

GoogleMAC's getMACDetails() delivers the post, and returns the JSON reply as a string:

```
// global
private static final String SITE_ADDR =
                       "http://www.google.com/loc/json";


private static String getMACDetails(String macAddress,
                                        int signalStrength)
{
  String data = "{\"version\":\"1.1.0\"," +
                   "\"request_address\":true," +
                   "\"wifi_towers\":
                       [{\"mac_address\":\"" + macAddress +
               "\",\"signal_strength\":" + signalStrength + "}]}";
  try {
    URL url = new URL(SITE_ADDR);
    URLConnection conn = url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);

    // write out JSON query parameters
    OutputStreamWriter writer =
          new OutputStreamWriter(conn.getOutputStream());
    writer.write(data);
    writer.flush();
    writer.close();

    // read the response
    BufferedReader in = new BufferedReader(
          new InputStreamReader(conn.getInputStream()));
    StringBuilder respStr = new StringBuilder();
    String line;
    while ((line = in.readLine()) != null)
      respStr.append(line);
    in.close();

    return respStr.toString();
  }
  catch (MalformedURLException e)
  {  System.out.println("URL not understood"); }
  catch (IOException e)
  { System.out.println("I/O Error");  }

  return null;
}   // end of getMACDetails()
```

The format of the returned data is:

```
{
  "access_token": "XXXX",
  "location": {
    "accuracy": 150,
    "address": {
      "city": "Kho Hong",
      "country": "Thailand",
      "country_code": "TH",
      "county": "Hat Yai",
      "region": "Songkhla"
    },
    "latitude": 7.0073644,
    "longitude": 100.5012336
  }
}
```

showResponse() saves the JSON response in "temp.json" for future reference, and calls showLatLong() and showAddress() to extract the latitude, longitude, and geographical address from the "location" tuple:

```
private static void showResponse(String respStr)
{
  if (respStr == null)
    System.out.println("No response received");
  else {
    try {
      JSONObject json = new JSONObject(respStr);
      WebUtils.saveString("temp.json", json.toString(2) );
                                 // indent the string
      JSONObject jLoc = json.getJSONObject("location");
      showLatLong(jLoc);
      showAddress(jLoc);
    }
    catch(JSONException e)
    {  System.out.println(e);  }
  }
}  // end of showResponse()
```

showLatLong() prints the contents of the "latitude", "longitude", and "accuracy" fields.

```
private static void showLatLong(JSONObject jLoc)
{
  try {
    double lat = Double.parseDouble( getString(jLoc, "latitude"));
    double lon = Double.parseDouble( getString(jLoc, "longitude"));
    System.out.printf("(lat, long): (%.6f, %.6f)\n", lat, lon);
    String accuracy = getString(jLoc, "accuracy");
    if (accuracy != null)
      System.out.println("Accuracy: " + accuracy);
  }
  catch(NumberFormatException e)
  {  System.out.println("No latitude and longitude found"); }
}  // end of showLatLong()
```

There's no real need to convert the latitude and longitude to numbers, but it allows them to be more easily formatted. Also, the "accuracy" field may not be present in the JSON reply.

showAddress() tries to print the address nicely spread over three lines, by examining the fields inside the "address" tuple:

```
private static void showAddress(JSONObject jLoc)
{
  JSONObject jAddr = null;
  try {
    jAddr = jLoc.getJSONObject("address");
  }
  catch(JSONException e){}

  if (jAddr == null)
    System.out.println("No address found");
  else {
    System.out.println("Address:");
    // 1st line
    System.out.print("  ");
    String streetNum = getString(jAddr, "street_number");
    if (streetNum != null)
      System.out.print(streetNum + " ");
    System.out.println(getString(jAddr, "street"));

    // 2nd line
    System.out.print("  ");
    String city = getString(jAddr, "city");
    if (city != null)
      System.out.print(city + ", ");

    String region = getString(jAddr, "region");
    if (region != null)
      System.out.print(region + ", ");

    System.out.println( getString(jAddr, "postal_code"));

    // 3rd line
    System.out.print("  ");
    String country = getString(jAddr, "country");
    if (country != null)
      System.out.print(country + ", ");

    System.out.println( getString(jAddr, "country_code"));
  }
}  // end of showAddress()
```

4.2. Skyhooking MAC addresses

The Skyhook location service (http://www.skyhookwireless.com/) is accessed via a HTTPS POST request containing an XML query with a MAC address and signal strength. As with Google, multiple addresses and strengths can be sent in order to improve the result. Latitude and longitude information are returned, and perhaps geographical information

Before I begin a detailed explanation, I should own up to always having received "Unable to determine location" error responses from Skyhook. I believe this is due to my location in Thailand being outside Skyhook's coverage area (http://www.skyhookwireless.com/howitworks/coverage.php). Their map does have some "blue spots" in my home town, Hat Yai, but I can't zoom in on the map sufficiently to determine where!

I started by downloading Skyhook's SDK for Windows XP SP3 from http://www.skyhookwireless.com/developers/sdk.php. It contains a test program, wpsapitest.exe, which successfully calculates a location based on my IP address, but returns "WPS_location failed (6)" when it tries a MAC address lookup. After examining the debug output from wpsapitext.exe stored in wpslog.txt, I confirmed that the SDK is successfully collecting details about nearby APs, and posting them to Skyhook's Web service in a XML query, but no location data is returned. wpsapitest.exe utilizes a Skyhook Wireless WiFi service (that's part of the SDK download), which needs to be installed on the machine.

Details on how to communicate with the Skyhook Web service are explained in the blog posts by coderrr at http://coderrr.wordpress.com/2008/09/10/get-the-physical-location-of-wireless-router-from-its-mac-address-bssid/ and Attack Vector at http://www.attackvector.org/geolocation-using-bssid/. Another good information source is the Skyhook developers network at http://groups.google.com/group/skyhook.

My SkyhookMac.java program builds a XML query which contains a single MAC address and signal strength read from the command line, combined with username and realm IDs. I created these IDs when I registered for the SDK. I double-checked the query format by looking at the log data for wpsapitext.exe.

```
// globals
private static final int DUMMY_STRENGTH = -75;
    // a strong signal, which means the AP is nearby

// change the following according to your Skyhook registration
private static final String USER_NAME = "????";
private static final String REALM = "????";


public static void main(String[] args)
{
  String macAddress = null;
  int signalStrength = DUMMY_STRENGTH;
  if (args.length == 1)
    macAddress = args[0];
  else if (args.length == 2) {
    macAddress = args[0];
    signalStrength = Integer.parseInt(args[1]);
  }
  else {
    System.out.println("Usage: java SkyhookMAC <MAC address>
                                                     [<signal>]");
    return;
  }

  String xmlQuery = "<?xml version='1.0'?>\n" +
       "<LocationRQ xmlns='http://skyhookwireless.com/wps/2005'" +
            " version='2.10' street-address-lookup='full'>\n" +
            "<authentication version='2.0'>\n" +
              "<simple>\n" +
                "<username>" + USER_NAME + "</username>\n" +
                "<realm>" + REALM + "</realm>  " +
              "</simple>\n" +
            "</authentication>  \n" +
            "<access-point>  \n" +
              "<mac>" +  macAddress + "</mac>  \n" +
              "<signal-strength>" + signalStrength +
                                       "</signal-strength>\n" +
              "<age>7984</age>\n" +
            "</access-point>  \n" +
       "</LocationRQ>";

  String respStr = getResponse(xmlQuery);
  if (respStr == null)
    System.out.println("No response received");
  else
    extractDetails(respStr);
}  // end of main()
```

It's possible to include multiple "access-point" tags in the XML query, and to switch off the retrieval of a geographical address.

Posting the query is a matter of opening an output stream to the Web server, and printing the XML string to it. The Skyhook server utilizes a HTTPS address, and so an HttpsURLConnection must be established. Also, the content type of the message must be XML rather than text. getResponse() implements these features:

```
// global
private static final String SITE_ADDR =
           "https://api.skyhookwireless.com/wps2/location";


private static String getResponse(String xmlQuery)
{
  try {
    URL url = new URL(SITE_ADDR);
    // open SSL https connection
    HttpsURLConnection conn =
            (HttpsURLConnection) url.openConnection();

    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestProperty("Content-Type", "text/xml"); // XML info

    // POST the XML query
    PrintWriter output = new PrintWriter(
       new OutputStreamWriter(conn.getOutputStream()));
    output.println(xmlQuery);
    output.flush();
    output.close();
    conn.connect();

    // read the response
    BufferedReader in = new BufferedReader(
              new InputStreamReader(conn.getInputStream()));
    StringBuilder respStr = new StringBuilder();
    String line;
    while ((line = in.readLine()) != null)
      respStr.append(line);
    in.close();
    return respStr.toString();
  }
  catch (MalformedURLException e)
  {  System.out.println("URL not understood"); }
  catch (IOException e)
  { System.out.println("I/O Error");  }

  return null;
}   // end of getResponse()
```

There are two basic response formats. A successful lookup will look something like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LocationRS version="2.10"
            xmlns="http://skyhookwireless.com/wps/2005">
  <location nap="1">
    <latitude>7.199997</latitude>
    <longitude>100.600006</longitude>
    <hpe>150</hpe>
  </location>
</LocationRS>
```

The response may also include address information; see http://www.attackvector.org/geolocation-using-bssid/ for an example.

The failure format, which I always receive, is:

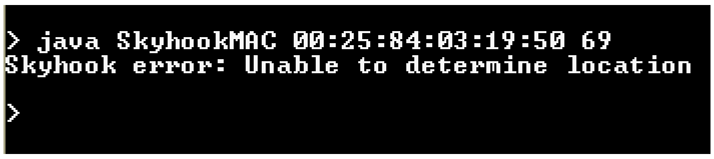
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LocationRS version="2.10"
            xmlns="http://skyhookwireless.com/wps/2005">
   <error>Unable to determine location</error>
</LocationRS>
```

The easiest way to access a response's XML data is with XPath queries (e.g. "/LocationRS/location/latitude" to get the latitude value). There may be multiple "location" nodes (including none), so I use a separate XPath query to get a set of all the location nodes, and then access the latitude and longitude of the first node (if it exists).

```
private static void extractDetails(String respStr)
// extract the latitude and longitude, or report the error;
// I've only been able to test the "/LocationRS/error" code branch
{
  try {
    InputSource is = new InputSource(new StringReader(respStr));
    Document respDoc = DocumentBuilderFactory.newInstance().
                                newDocumentBuilder().parse(is);
    XPath xPath = XPathFactory.newInstance().newXPath();

    // Get the Location nodes using XPath
    NodeList nodes = (NodeList)xPath.evaluate("/LocationRS/Location",
                              respDoc, XPathConstants.NODESET);
    int nodeCount = nodes.getLength();
    if (nodeCount != 0) {
      System.out.println("No. of location nodes: " + nodeCount);

      // only report the first latitude and longitude
      String lat = (String)xPath.evaluate("latitude",
                           nodes.item(0), XPathConstants.STRING);
      String lon = (String)xPath.evaluate("longitude",
                           nodes.item(0), XPathConstants.STRING);
      if (lat == null)
        System.out.println("No latitude and longitude found");
      else
        System.out.println("First (lat,lon): (" +
                                   lat + ", " + lon + ")");
    }
    else {
      String errorMsg = (String)xPath.evaluate("/LocationRS/error",
                             respDoc, XPathConstants.STRING);
      if (errorMsg != null)
        System.out.println("Skyhook error: " + errorMsg);
      else
        System.out.println("Could not parse the response string");
    }
  }
  catch(Exception e)
  {  System.out.println(e); }
}  // end of extractDetails()
```

The error message can be retrieved with a "/LocationRS/error" XPath.

Figure 10 shows the output when I run SkyhookMAC.java.



```
> java SkyhookMAC 00:25:84:03:19:50 69
Skyhook error: Unable to determine location

>
```

Figure 10. Using Skyhook Wireless to Lookup a MAC address.

5. Using the Latitude and Longitude

The latitude and longitude information returned by GoogleMAC.java and SkyhookMAC.java can be used for mapping, but I talked about that in the last chapter, so won't revisit it here. Instead, I'll find a geographical address for a given latitude and longitude, a task called *reverse geocoding*.

GoogleMAC.java gave an impressively accurate latitude and longitude but a disappointing address (no street, city, or zip code). Naturally, the quality of the results will vary, but how can poor results be improved? Figure 2 shows that I have three programs for digging out address details: GoogleLatLong.java, OSMLatLong.java, and YahooLatLong.java which (as the names imply) employ Web services from Google, OSM, and Yahoo respectively.

All three programs have a similar structure: a HTTP GET request containing the latitude and longitude is sent to the service, and back comes a JSON-formatted response containing the address. The services mainly differ in their terms of use (for example, Google's geocoding API has a query limit of 2,500 requests per day; and can only be used in conjunction with a Google map). Also, the quality of the results varies quite a bit, but that may just be that the south of Thailand isn't as well covered as the US and Europe.

I'll explain the GoogleLatLong.java code in detail, but only sketch out the query and response formats for the OSM and Yahoo services, due to their coding similarities.

5.1. Googling a Geographical Location

Reverse geocoding with Google is explained at http://code.google.com/apis/maps/documentation/geocoding/#ReverseGeocoding.

A GET request is sent to the service with the latitude and longitude assigned to a "latlng" parameter. For instance, the following delivers my latitude/longitude in Hat Yai:

```
http://maps.googleapis.com/maps/api/geocode/json?
            latlng=7.0073644,100.5012336&sensor=false
```

The "sensor" parameter indicates whether the coordinates came from a GPS sensor (they didn't).

The returned JSON result is normally quite lengthy, containing several alternative answers which specify the address in terms of region, city, neighborhood, street address, postal code, and mixes of those elements. For example, the first answer tuple for the above query is:

```
{
  "status": "OK",
  "results": [ {
    "types": [ "locality", "political" ],
    "formatted_address": "Kho Hong, Hat Yai, Songkhla, Thailand",
    "address_components": [ {
      "long_name": "Kho Hong",
      "short_name": "Kho Hong",
      "types": [ "locality", "political" ]
    }, {
      "long_name": "Hat Yai",
      "short_name": "Hat Yai",
      "types": [ "administrative_area_level_3", "political" ]
    }, {
      "long_name": "Songkhla",
      "short_name": "Songkhla",
      "types": [ "administrative_area_level_1", "political" ]
    }, {
      "long_name": "Thailand",
      "short_name": "TH",
      "types": [ "country", "political" ]
    } ],
    "geometry": {
      "location": {
        "lat": 7.0016937,
        "lng": 100.4996926
      },
      "location_type": "APPROXIMATE",
      "viewport": {
        "southwest": {
          "lat": 6.9573925,
          "lng": 100.4356629
        },
        "northeast": {
          "lat": 7.0459907,
          "lng": 100.5637223
        }
      },
      "bounds": {
        "southwest": {
          "lat": 6.9688668,
          "lng": 100.4608475
        },
        "northeast": {
          "lat": 7.0543579,
          "lng": 100.5391657
        }
      }
    }
  },
  // four more result tuples
]}
```

A simple way of summarizing all the answers is to extract each tuple's "formatted_address" field. All the values returned by the previous query are shown below:

```
"formatted_address": "Kho Hong, Hat Yai, Songkhla, Thailand"
"formatted_address": "Nam Noi, Hat Yai, Songkhla 90110, Thailand"
"formatted_address": "Hat Yai, Songkhla, Thailand"
"formatted_address": "Songkhla, Thailand"
"formatted_address": "Thailand"
```

Generally, the addresses are ordered from most to least specific, with the more exact being the first. But, as the above example shows, the other addresses may contain useful additional information. Collectively, these addresses are an improvement over GoogleMAC's output (see Figure 8), since they include the city name and zip code. However, there's no street address, or indication that the coordinate is inside Prince of Songkla University.

Getting a Response

GoogleLatLong.java's getResponse() method sends a GET request including a latitude and longitude, and returns the JSON response as a string.

```
// global
```

```
private static final String SITE_ADDR =
             "http://maps.google.com/maps/api/geocode/json";


private static String getResponse(double lat, double lon)
{
  String coords = "" + lat + ", " + lon;
  BufferedReader rd = null;
  try {
    URL url = new URL(SITE_ADDR + "?address=" +
                URLEncoder.encode(coords, "UTF-8") +
                   "&sensor=false");
    HttpURLConnection conn =
            (HttpURLConnection) url.openConnection();
    conn.connect();
    rd = new BufferedReader(
       new InputStreamReader(conn.getInputStream()));

    StringBuilder respStr = new StringBuilder();
    String line;
    while ((line = rd.readLine()) != null)
      respStr.append(line);
    rd.close();

    return respStr.toString();
  }
  catch (MalformedURLException e)
  {  System.out.println("URL not understood"); }
  catch (IOException e)
  { System.out.println("I/O Error");  }

  return null;
}  // end of getResponse()
```

Printing the Response

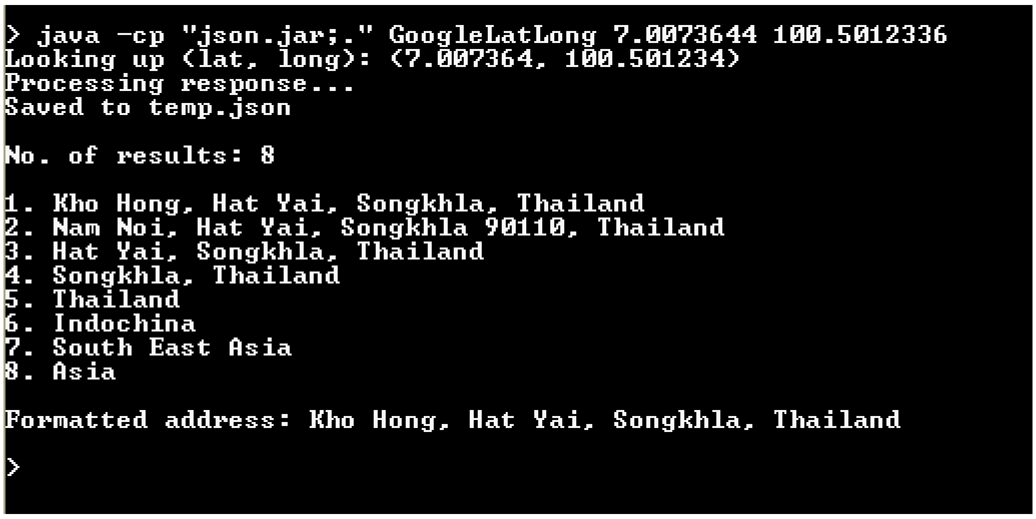
The JSON data is saved into a file, and also passed to extractAddress() which returns the first "formatted_address" value.

```
private static void showResponse(String respStr)
{
  if (respStr == null)
    System.out.println("No response received");
  else {
    System.out.println("Processing response...");
    try {
      JSONObject json = new JSONObject(respStr);
      WebUtils.saveString("temp.json", json.toString(2) );

      String addr = extractAddress(json);
      if (addr != null)
        System.out.println("\nFormatted address: " + addr);
      else
        System.out.println("No address found");
    }
    catch(JSONException e)
    {  System.out.println(e);  }
  }
}  // end of showResponse()
```

extractAddress() treats the "results" tuples as an array, iterating through them, printing every "formatted_address" value. The first one is returned.

```
private static String extractAddress(JSONObject json)
{
  try {
    JSONArray jResults = json.getJSONArray("results");
    int numMatches = jResults.length();
    System.out.println("\nNo. of results: " + numMatches + "\n");
    if (numMatches == 0)
      return null;

    JSONObject jRes;
    for (int i = 0; i < numMatches; i++) {
      jRes = jResults.getJSONObject(i);
      System.out.println((i+1) + ". " +
                       jRes.getString("formatted_address"));
    }
    return jResults.getJSONObject(0).
                  getString("formatted_address");
  }
  catch(Exception e)
  {  System.out.println(e);  }

  return null;
}  // end of extractAddress()
```

The execution of GoogleLatLong.java with my latitude and longitude is shown in Figure 11.



```
> java -cp "json.jar;." GoogleLatLong 7.0073644 100.5012336
Looking up (lat, long): (7.007364, 100.501234)
Processing response...
Saved to temp.json

No. of results: 8

1. Kho Hong, Hat Yai, Songkhla, Thailand
2. Nam Noi, Hat Yai, Songkhla 90110, Thailand
3. Hat Yai, Songkhla, Thailand
4. Songkhla, Thailand
5. Thailand
6. Indochina
7. South East Asia
8. Asia

Formatted address: Kho Hong, Hat Yai, Songkhla, Thailand

>
```

Figure 11. Using Google Reverse Geocoding.

5.2. Using OSM to Find a Geographical Location

Reverse geocoding in OSM is implemented by its Nominatim tool (http://nominatim.openstreetmap.org), which is described at http://wiki.openstreetmap.org/wiki/Nominatim.

A typical GET query:

```
http://nominatim.openstreetmap.org/reverse?format=json&
                   lat=7.0073644&lon=100.5012336&
                   zoom=18&addressdetails=1
```

The zoom value determines the level of detail returned, where 0 is at the country level and 18 is at a house/building scale.

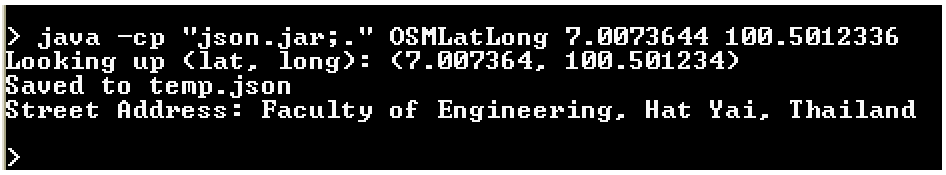
The JSON result is:

```
{
  "address": {
    "city": "Hat Yai",
    "country": "Thailand",
    "country_code": "th",
    "university": "Faculty of Engineering"
  },
  "category": "amenity",
  "display_name": "Faculty of Engineering, Hat Yai, Thailand",
  "licence": "Data Copyright OpenStreetMap Contributors,
             Some Rights Reserved. CC-BY-SA 2.0.",
  "osm_id": "767329196",
  "osm_type": "node",
  "place_id": "70817374",
  "type": "university"
}
```

As with Google, a large range of fields may contain data, but the "display_name" field holds a nicely formatted string (in a similar way to Google's "formatted_address"). Unlike Google, only a single address is returned.

OSM has a less restrictive usage policy than Google, but encourages heavy users to include an "email" key/value pair in the query so that OSM can contact the sender if their load becomes too severe (see http://wiki.openstreetmap.org/wiki/Nominatim_usage_policy for details).

The execution of OSMLatLong.java with my latitude and longitude is shown in Figure 12.



```
> java -cp "json.jar;." OSMLatLong 7.0073644 100.5012336
Looking up (lat, long): (7.007364, 100.501234)
Saved to temp.json
Street Address: Faculty of Engineering, Hat Yai, Thailand

>
```

Figure 12. Using OSM Reverse Geocoding.

Unlike Google, OSM identified my location as a university (in the "type" field), but this isn't included in the "display_name" string, which only mentions the faculty. The result is impressive because the engineering faculty is very close to my office, and is the large building shown in Figure 9.

5.3. Using Yahoo to Find a Geographical Location

Reverse geocoding in Yahoo is available through its PlaceFinder service, http://developer.yahoo.com/geo/placefinder/, which replaced the older Yahoo Maps Web Services Geocoding API. There's a user guide at http://developer.yahoo.com/geo/placefinder/guide, and a developers forum at http://developer.yahoo.net/forum/?showforum=124.

Yahoo requires users to sign up for an application ID, but it's free. There's also a query limit, but it's a generous 50,000 requests per day.

A typical GET query:

```
http://where.yahooapis.com/geocode?q=7.0073644,+100.5012336&
                   gflags=R&appid=[your_application_ID]
```

The "gflags=R" value specifies that reverse geocoding is required.

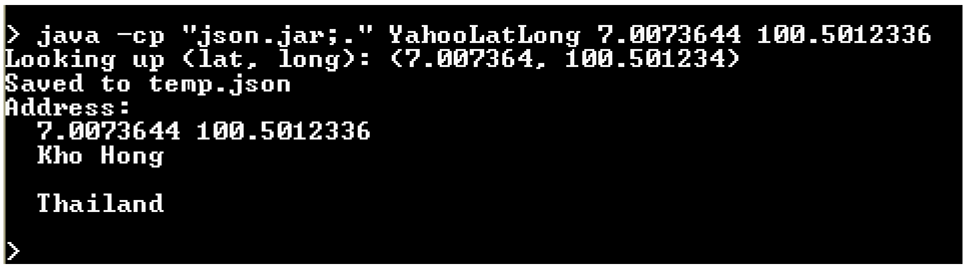
The JSON result is:

```
{"ResultSet": {
  "Error": 0,
  "ErrorMessage": "No error",
  "Found": 1,
  "Locale": "us_US",
  "Quality": 99,
  "Results": [{
    "city": "Kho Hong",
    "country": "Thailand",
    "countrycode": "TH",
    "county": "Hat Yai",
    "countycode": "",
    "hash": "",
    "house": "",
    "latitude": "7.007364",
    "line1": "7.0073644 100.5012336",
    "line2": "Kho Hong",
    "line3": "",
    "line4": "Thailand",
    "longitude": "100.501234",
    "name": "7.0073644 100.5012336",
    "neighborhood": "",
    "offsetlat": "7.007364",
    "offsetlon": "100.501234",
    "postal": "",
    "quality": 99,
    "radius": 500,
    "state": "Songkhla",
    "statecode": "",
    "street": "",
    "unit": "",
    "unittype": "",
    "uzip": null,
    "woeid": 1207033,
    "woetype": 7,
    "xstreet": ""
  }],
  "version": "1.0"
}}
```

There's no single field which contains all the address data, although the combined "line1", "line2", "line3", and "line4" fields come close.

The Where On Earth ID (WOEID) returned by the service (e.g. 1207033 in the example above) can be passed to Yahoo's GeoPlanet API to potentially obtain more geographical information (see http://developer.yahoo.com/geo/geoplanet/guide/), but I didn't investigate that approach.

The execution of YahooLatLong.java with my PSU latitude and longitude is shown in Figure 13.



```
> java -cp "json.jar;." YahooLatLong 7.0073644 100.5012336
Looking up (lat, long): (7.007364, 100.501234)
Saved to temp.json
Address:
  7.0073644 100.5012336
  Kho Hong

  Thailand

>
```

Figure 13. Using Yahoo Reverse Geocoding.

Yahoo's result is disappointing because it doesn't include a street address, state, or zip code. State information is available in the JSON response, but isn't included in the "line" fields printed by YahooLatLong.

6. Using IP addresses

Now it's the turn of IP addresses for finding latitudes, longitudes, and geographical addresses. They're just as useful as MACs, so long as they're static rather than dynamic or private. Static addresses are uniquely assigned to a particular computer or device, and so are more likely to have location information stored somewhere.

Dynamic IP addressing is typically used by ISPs to let multiple users share a limited set of addresses. The address is dynamic in the sense that it's only assigned to the user's device for the duration of that Internet session or other limited time. Once the user disconnects, their address goes back into a pool so it can be re-assigned.

Private IP addresses are only unique locally, such as in a company or university. When devices with private IP addresses need to communicate globally, they must undergo network address translation (NAT). There are three ranges of IPv4 addresses reserved for private networks: 10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255.

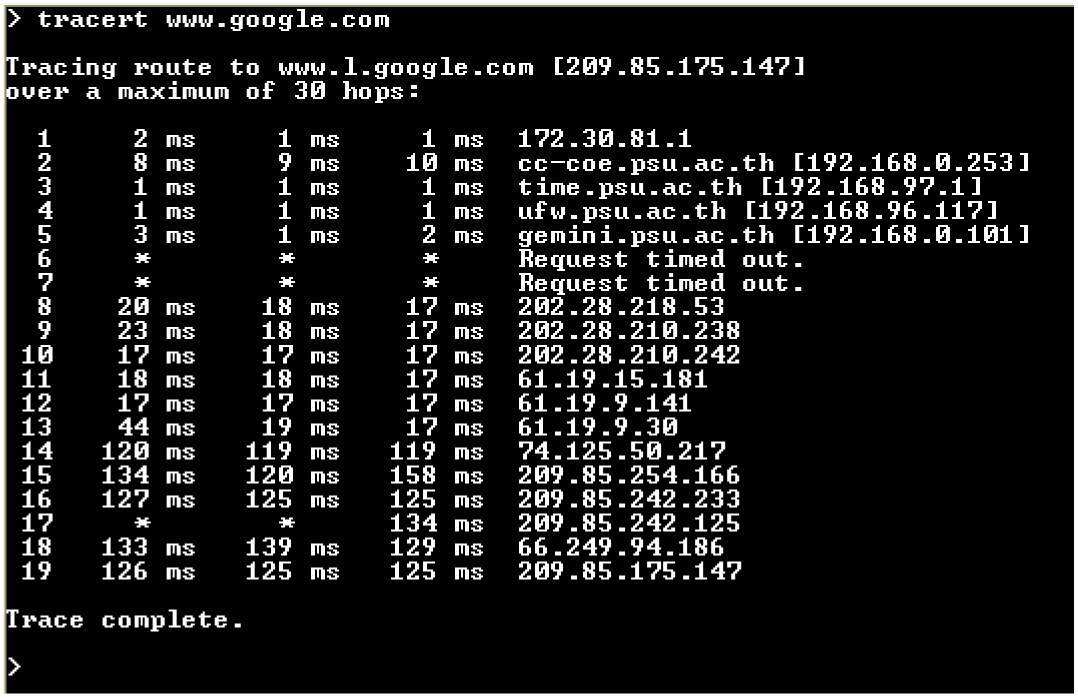
When I ran AddrViewer at the start of this chapter, my netbook's IP address was given as 172.30.81.158, a private departmental address (see Figure 3). I need to find a static IP address, sometimes called an outward-facing or external address, because it can be used globally.

There are several Web services that can lookup a user's external IP address, starting from information automatically included in the GET request sent to them. Some results for my netbook are shown in Table 1.

Service	External IP Address
WhatIP (http://www.whatip.com/)	202.12.74.4
DynDNS (http://checkip.dyndns.org/)	202.12.73.65
WebIP (http://webipaddress.net/)	202.12.74.4

Table 1. IP Lookup Service Results.

Another way of obtaining an external IP is to use traceroute (tracert on Windows). It sends a sequence of ICMP packets to a destination host, tracing out the routers that it passes through. On my netbook, tracert produces the results shown in Figure 14.



```
> tracert www.google.com

Tracing route to www.l.google.com [209.85.175.147]
over a maximum of 30 hops:

  1     2 ms     1 ms     1 ms  172.30.81.1
  2     8 ms     9 ms    10 ms  cc-coe.psu.ac.th [192.168.0.253]
  3     1 ms     1 ms     1 ms  time.psu.ac.th [192.168.97.1]
  4     1 ms     1 ms     1 ms  ufw.psu.ac.th [192.168.96.117]
  5     3 ms     1 ms     2 ms  gemini.psu.ac.th [192.168.0.101]
  6     *        *        *     Request timed out.
  7     *        *        *     Request timed out.
  8    20 ms    18 ms    17 ms  202.28.218.53
  9    23 ms    18 ms    17 ms  202.28.210.238
 10    17 ms    17 ms    17 ms  202.28.210.242
 11    18 ms    18 ms    17 ms  61.19.15.181
 12    17 ms    17 ms    17 ms  61.19.9.141
 13    44 ms    19 ms    17 ms  61.19.9.30
 14   120 ms   119 ms   119 ms  74.125.50.217
 15   134 ms   120 ms   158 ms  209.85.254.166
 16   127 ms   125 ms   125 ms  209.85.242.233
 17     *        *      134 ms  209.85.242.125
 18   133 ms   139 ms   129 ms  66.249.94.186
 19   126 ms   125 ms   125 ms  209.85.175.147

Trace complete.

>
```

Figure 14. Tracert from My Netbook to Google.

The first few hops are between machines within the department until hop 8 which passes through the external IP address 202.28.218.53. This isn't a great result because it resolves to UniNet, an inter-university network in Bangkok, 750 km away. I know that by querying WebIP with http://webipaddress.net/ip/202.28.218.53 (see section 6.2. below).

6.1. Looking up an IP address with DynDNS

The DynDNS server (http://checkip.dyndns.com/) returns a simple bit of HTML containing the detected external IP address. For example:

```
<html>
<head><title>Current IP Check</title></head>
<body>Current IP Address: 202.12.73.65</body>
</html>
```

DetectIP.java duplicates this by sending a GET request to http://checkip.dyndns.com/, and processes the response as a string. It's extractData() method strips away the HTML tags, and returns the text after the ":":

```
private static String extractData(String respStr)
{
  String resStr = respStr.replaceAll("\\<.*?>","");  // remove tags

  int startPosn = resStr.indexOf(':');
  if (startPosn == -1)
    return resStr;
  else
    return resStr.substring(startPosn+2).trim();
}  // end of extractData()
```

The IP address returned by DynDNS, 202.12.73.65, can be looked up using WebIP (by loading http://webipaddress.net/ip/202.12.73.65 into a browser). However, we can query WebIP directly without involving DynDNS, as explained in the next section. Another approach is to send the IP address to a Whois service, which is the topic of section 7.

6.2. Retrieving More than an IP Address

Once contacted, the WebIP service returns my external IP address, a latitude and longitude, a geographical address, and even a map! The GET request is sent to http://webipaddress.net/what-is-my-ip-address, and a rather complex page of HTML is returned It's also possible to ask for details about a specific address by contacting http://webipaddress.net/what-is-my-ip-address/ip/<the IP address>.

The retrieved details are presented to us in a HTML table separated from the rest of the page by a DIV tag, <div class="ipdetail">. For example:

```
<div class="ipdetail">
<table width="380" cellpadding="0" cellspacing="6">
   <tr>
     <td valign="top"><strong>IP Information:</strong></td>
     <td width="223"><strong>202.12.74.4</strong></td>
   </tr>
   <tr>
     <td valign="top"><strong>ISP:</strong></td>
     <td>Prince of Songkla University</td>
   </tr>
   <tr>
     <td valign="top"><strong>Organization:</strong></td>
     <td>Prince of Songkla University</td>
   </tr>
   <tr>
     <td valign="top"><strong>City:</strong></td>
     <td>Songkhla</td>
```

```
    </tr>
    <tr>
      <td valign="top"><strong>Region / State:</strong></td>
      <td>68 Songkhla</td>
    </tr>
    <tr>
      <td valign="top"><strong>Country:</strong></td>
      <td>Thailand <img src='flag/th.gif' width='16'
             height='11' title='Thailand' alt='TH'></img></td>
    </tr>
    <tr>
      <td valign="top"><strong>Country Code:</strong></td>
      <td>TH</td>
     </tr>
     <tr>
       <td width="115" valign="top"><strong>Continent:</strong></td>
       <td>AS</td>
     </tr>
     <tr>
       <td><strong>Time Zone:</strong></td>
       <td>Asia/Bangkok</td>
     </tr>
     <tr>
       <td><strong>Latitude:</strong></td>
       <td>7.1999998092651</td>
     </tr>
     <tr>
       <td><strong>Longitude:</strong></td>
       <td>100.59999847412</td>
     </tr>
     <tr>
       <td><strong>Postal Code:</strong></td>
       <td>n/a</td>
     </tr>
     <tr>
       <td><strong>Metro Code:</strong></td>
       <td>n/a</td>
     </tr>
     <tr>
       <td><strong>Area Code:</strong></td>
       <td>n/a</td>
     </tr>
  </table>
</div>
```

The getResponse() method in DetailsUsingIP.java sends a GET request, then returns the table as a single, long string:

```
// global
private static final String SITE_ADDR =
           "http://webipaddress.net/what-is-my-ip-address";


private static String getResponse(String ipAddress)
{
  try {
    URL url = null;
    if (ipAddress == null)
      url = new URL(SITE_ADDR);   // lookup own IP address
    else
```

```
      url = new URL(SITE_ADDR + "/ip/" + ipAddress);
                 // lookup address supplied on the command line

    URLConnection conn = url.openConnection();
    BufferedReader rd = new BufferedReader(
             new InputStreamReader(conn.getInputStream()));

    boolean usefulInfo = false;
    StringBuilder tableStr = new StringBuilder();
    String line;
    while ((line = rd.readLine()) != null) {
      if (line.startsWith("<div class=\"ipdetail\">"))
        usefulInfo = true;
      if ((line.startsWith("</div>")) && usefulInfo)
        break;    // finish processing

      if (usefulInfo)
        tableStr.append(line.trim() + "\n");
    }
    return tableStr.toString();
  }
  catch (MalformedURLException e)
  {  System.out.println(e); }
  catch (IOException e)
  {  System.out.println(e); }

  return null;
}  // end of getResponse()
```

extractData() reformats the HTML table into key:value text lines via a series of regular expression transformations. For instance, the table from above will become:

```
IP Information: 202.12.74.4
ISP: Prince of Songkla University
Organization: Prince of Songkla University
City: Songkhla
Region / State: 68 Songkhla
Country: Thailand
Country Code: TH
Continent: AS
Time Zone: Asia/Bangkok
Latitude: 7.1999998092651
Longitude: 100.59999847412
Postal Code: n/a
Metro Code: n/a
Area Code: n/a
```

The code for extractData() is:

```
private static String extractData(String respStr)
{
  String tableData = respStr.replaceAll("\\<.*?>",""); //remove tags

  tableData = tableData.replaceAll("(\\s){2}","$1");
      // replace two consecutive white spaces with a single space

  tableData = tableData.replaceAll(":\\n", ": ");
      // combine key and value lines
  tableData = tableData.replaceAll("\\n\\n", "\n").trim();
     // remove blank lines

  return tableData;
}  // end of extractData()
```

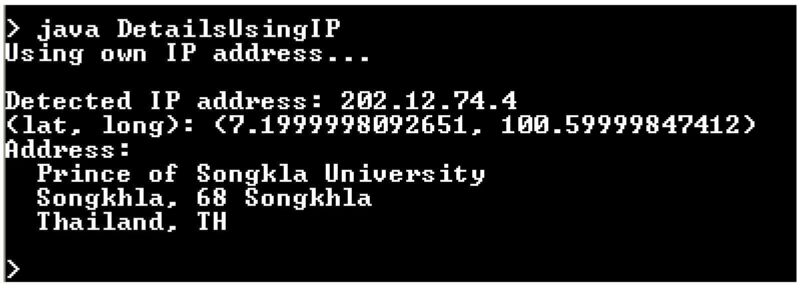
The key:value pairs are stored in a HashMap by buildMap():

```
private static HashMap<String,String> buildMap(String tableData)
{
  String[] lines = tableData.split("\\n");
  int numLines = lines.length;
  if (numLines > 0) {
    HashMap<String,String> map = new HashMap<String,String>();
    String[] pair;
    for (int i=0; i < numLines; i++) {
      pair = lines[i].split(":");
      if (pair.length != 2)
        System.out.println("Problem parsing table line: \"" +
                                        lines[i] + "\"");
      else
        map.put(pair[0].trim(), pair[1].trim());  // key:value pairs
    }
    return map;
  }
  return null;
}  // end of buildMap()
```

showResponse() illustrates how to build and use the HashMap:

```
private static void showResponse(String respStr)
/* print some of the response data after converting it to
   a HashMap of key:value pairs
*/
{
  if (respStr == null)
    System.out.println("No address found");
  else {
    String tableData = extractData(respStr);
    HashMap<String,String> map = buildMap(tableData);
    if (map != null) {
      System.out.println("\nDetected IP address: " +
                             map.get("IP Information") );
      System.out.println("(lat, long): (" + map.get("Latitude") +
                    ", " + map.get("Longitude") + ")");
      System.out.println("Address:");
      System.out.println("  " + map.get("Organization"));
      System.out.println("  " + map.get("City") + ", " +
                                map.get("Region / State") );
      System.out.println("  " + map.get("Country") + ", " +
                                map.get("Country Code") );
    }
  }
}  // end of showResponse()
```

The output of DetailsUsingIP when applied to my netbook is shown in Figure 15.

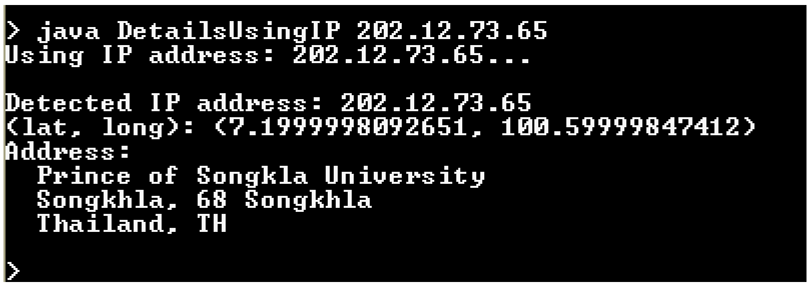


```
> java DetailsUsingIP
Using own IP address...

Detected IP address: 202.12.74.4
(lat, long): (7.1999998092651, 100.59999847412)
Address:
  Prince of Songkla University
  Songkhla, 68 Songkhla
  Thailand, TH

>
```

Figure 15. Details about my IP Address.

Although the address details are fairly accurate, there's no mention of the town (Hat Yai), and the latitude and longitude is a location in Songkla, the provincial capital, 20 km away. It's actually the same spot as shown in Figure 1, which I confirmed by entering the latitude and longitude values into Google Maps.

The DynDNS service from the last section returned a different IP address (202.12.73.65), but WebIP resolves it to the same latitude, longitude and address as 202.12.74.4 (see Figure 16).



```
> java DetailsUsingIP 202.12.73.65
Using IP address: 202.12.73.65...

Detected IP address: 202.12.73.65
(lat, long): (7.1999998092651, 100.59999847412)
Address:
  Prince of Songkla University
  Songkhla, 68 Songkhla
  Thailand, TH

>
```

Figure 16. Details about the 202.12.73.65 Address.

7. Using Whois

The Whois protocol employs online databases of registered users of IP addresses. Unfortunately, a Whois query must be sent to the correct regional database in order to find the details, but there's a number of Web services that deal with this routing issue.

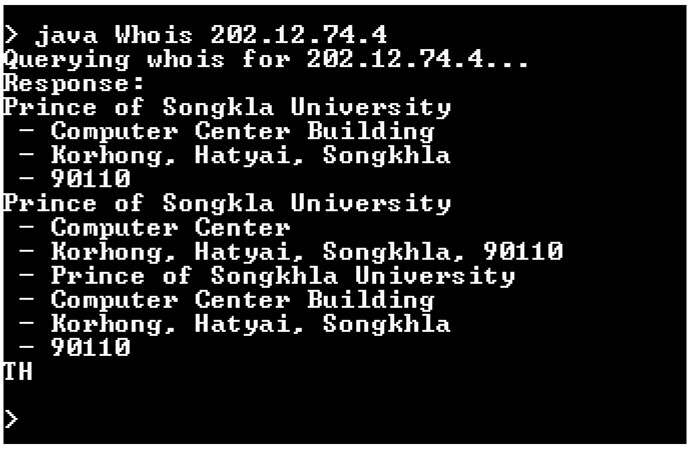
My Whois.java program posts a Whois query to http://uwhois.com/, which uses the included IP address to determine which Whois database to question. For example, asking about 202.12.74.4 (returned by WebIP in Figure 15) results in the following reply:

```
[whois.apnic.net]
% [whois.apnic.net node-5]
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:        202.12.74.0 - 202.12.74.255
netname:        MOR-OR-NET2
country:        TH
descr:          Prince of Songkla University
descr:          Computer Center Building
descr:          Korhong, Hatyai, Songkhla
descr:          90110
admin-c:        WW100-AP
```

```
tech-c:          SH72-AP
status:          ASSIGNED PORTABLE
mnt-by:          APNIC-HM
changed:         hm-changed@apnic.net 20050322
source:          APNIC

person:          Wiboon Warasittichai
nic-hdl:         WW100-AP
e-mail:          wiboon.w@psu.ac.th
address:         Prince of Songkla University
address:         Computer Center
address:         Korhong, Hatyai, Songkhla, 90110
phone:           +66-74-282128
fax-no:          +66-74-282111
country:         TH
mnt-by:          MAINT-AS9464
changed:         hm-changed@apnic.net 20050322
source:          APNIC

person:          Sakorn Hangsapruek
nic-hdl:         SH72-AP
e-mail:          sakorn.h@psu.ac.th
address:         Prince of Songkhla University
address:         Computer Center Building
address:         Korhong, Hatyai, Songkhla
address:         90110
phone:           +66-74-282097
fax-no:          +66-74-282111
country:         TH
mnt-by:          MAINT-AS9464
changed:         hm-changed@apnic.net 20050322
source:          APNIC
```

The information comes from APNIC, the Asia Pacific Network Information Center which holds IP registration details for my region (I'm based in Thailand). This output also shows that it's possible to receive multiple Whois records, especially if the IP address originates from a large organization, such as a university.

My Whois.java application treats this data as a large string, extracting all the lines that start with descr", "address", and "country". The program's execution is shown in Figure 17.

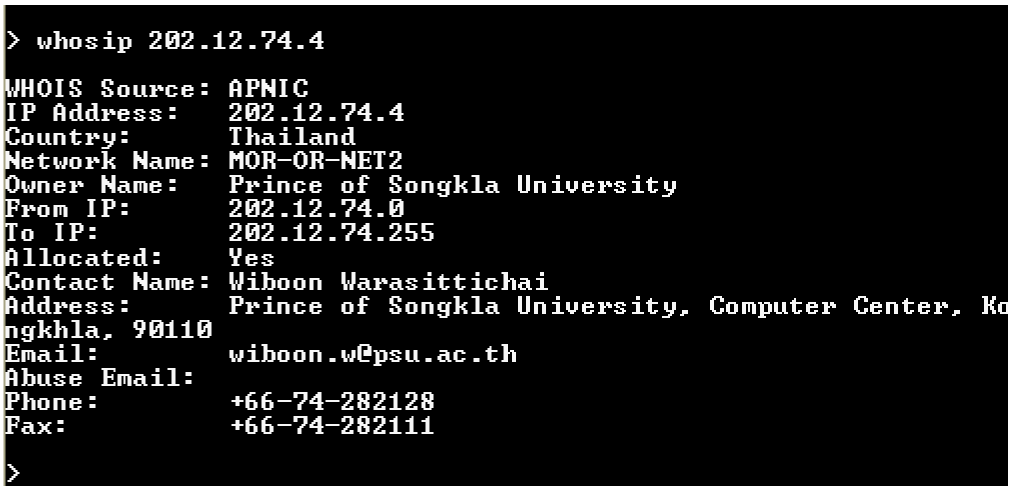


```
> java Whois 202.12.74.4
Querying whois for 202.12.74.4...
Response:
Prince of Songkla University
 - Computer Center Building
 - Korhong, Hatyai, Songkhla
 - 90110
Prince of Songkla University
 - Computer Center
 - Korhong, Hatyai, Songkhla, 90110
 - Prince of Songkhla University
 - Computer Center Building
 - Korhong, Hatyai, Songkhla
 - 90110
TH

>
```

Figure 17. Whois.java Output.

The PSU computer center mentioned twice in Figure 17 is the main computing facility at my university.

An alternative to http://uwhois.com/ is to utilize a Windows whois tool, such as the one from Microsoft (http://technet.microsoft.com/en-us/sysinternals/bb897435.aspx) or NirSoft (http://www.nirsoft.net/utils/whosip.html). The execution of the NirSoft whosip.exe command is shown in Figure 18.



```
> whosip 202.12.74.4

WHOIS Source: APNIC
IP Address:   202.12.74.4
Country:      Thailand
Network Name: MOR-OR-NET2
Owner Name:   Prince of Songkla University
From IP:      202.12.74.0
To IP:        202.12.74.255
Allocated:    Yes
Contact Name: Wiboon Warasittichai
Address:      Prince of Songkla University, Computer Center, Ko
ngkhla, 90110
Email:        wiboon.w@psu.ac.th
Abuse Email:
Phone:        +66-74-282128
Fax:          +66-74-282111

>
```

Figure 18. NirSoft Whosip Output.

The whosip.exe command can be interfaced to Java using my SaferExec class, in a similar way to how WirelessNetView.exe was utilized by FindWifiMACs.java in section 3.2.