Using the Geocoder

Geocoding lets you translate between street addresses and longitude/latitude map coordinates. This can give you a recognizable context for the locations and coordinates used in location-based services and map-based Activities.

The Geocoder class provides access to two geocoding functions:

- **Forward Geocoding** Finds the latitude and longitude of an address.
- **Reverse Geocoding** Finds the street address for a given latitude and longitude.

The results from these calls will be contextualized using a locale, where a locale is used to defi ne your usual location and language. The following snippet shows how you set the locale when creating your Geocoder. If you don't specify a locale, it will assume your device's default.

```
Geocoder geocoder = new Geocoder(getApplicationContext(),
Locale.getDefault());
```

Both geocoding functions return a list of Address objects. Each list can contain several possible results, up to a limit you specify when making the call.

Each Address object is populated with as much detail as the Geocoder was able to resolve. This can include the latitude, longitude, phone number, and increasingly granular address details from country to street and house number.

Geocoder lookups are performed synchronously, so they will block the calling thread. For slow data connections, this can lead to an Application Unresponsive dialog. In most cases, it's good form to move these lookups into a Service or background thread, as shown in Chapter 8.

For clarity and brevity, the calls made in the code samples within this chapter are made on the main application thread.

Reverse Geocoding

Reverse geocoding returns street addresses for physical locations, specifi ed by latitude/longitude pairs. It provides a recognizable context for the locations returned by location-based services.

To perform a reverse lookup, you pass the target latitude and longitude to a Geocoder's getFromLocation method. It will return a list of possible matching addresses. If the Geocoder could not resolve any addresses for the specifi ed coordinate, it will return null.

The following example shows how to reverse-geocode your last known location:

```
location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
double latitude = location.getLatitude();
double longitude = location.getLongitude();
Geocoder gc = new Geocoder(this, Locale.getDefault());
List<Address> addresses = null;
try {
addresses = gc.getFromLocation(latitude, longitude, 10);
} catch (IOException e) {}
```

The accuracy and granularity of reverse lookups are entirely dependent on the quality of data in the geocoding database; as such, the quality of the results may vary widely between different countries and locales.

Forward Geocoding

Forward geocoding (or just geocoding) determines map coordinates for a given location.

What constitutes a valid location varies depending on the locale (geographic area) within which you're searching. Generally, it will include regular street addresses of varying granularity (from country to street name and number), postcodes, train stations, landmarks, and hospitals. As a general guide, valid search terms will be similar to the addresses and locations you can enter into the Google Maps search bar.

To do a forward-geocoding lookup, call getFromLocationName on a Geocoder instance. Pass in the location you want the coordinates for and the maximum number of results to return, as shown in the snippet below:

```
List<Address> result = geocoder.getFromLocationName(aStreetAddress, maxResults);
```

The returned list of Addresses can include multiple possible matches for the named location. Each address result will include latitude and longitude and any additional address information available for those coordinates. This is useful to confi rm that the correct location was resolved, as well as providing address specifi cs when searching for landmarks.

As with reverse geocoding, if no matches are found, null will be returned. The availability, accuracy, and granularity of geocoding results will depend entirely on the database available for the area you're searching.

When doing forward lookups, the Locale specifi ed when creating the Geocoder object is particularly important. The Locale provides the geographical context for interpreting your search requests, as the same location names can exist in multiple areas. Where possible, consider selecting a regional Locale to help avoid place name ambiguity.

Additionally, try to use as many address details as possible. For example, the following code snippet demonstrates a forward geocode for a New York street address:

```
Geocoder fwdGeocoder = new Geocoder(this, Locale.US);
String streetAddress = "160 Riverside Drive, New York, New York";
List<Address> locations = null;
try {
locations = fwdGeocoder.getFromLocationName(streetAddress, 10);
} catch (IOException e) {}
```

For even more specifi c results, use the getFromLocationName overload, that lets you restrict your search to within a geographical bounding box, as shown in the following snippet:

```
List<Address> locations = null;
try {
locations = fwdGeocoder.getFromLocationName(streetAddress, 10, n, e, s, w);
} catch (IOException e) {}
```

This overload is particularly useful in conjunction with a MapView as you can restrict the search to within the visible map.

Geocoding "Where Am I?"

Using the Geocoder, you can determine the street address at your current location. In this example, you'll further extend the "Where Am I?" project to include and update the current street address whenever the device moves.

Open the WhereAmI Activity. Modify the updateWithNewLocation method to instantiate a new Geocoder object, and call the getFromLocation method, passing in the newly received location and limiting the results to a single address.

Extract each line in the street address, as well as the locality, postcode, and country, and append this information to an existing Text View string.

```
private void updateWithNewLocation(Location location) {
String latLongString;
TextView myLocationText;
myLocationText = (TextView)findViewById(R.id.myLocationText);
String addressString = "No address found";
if (location != null) {
double lat = location.getLatitude();
double lng = location.getLongitude();
latLongString = "Lat:" + lat + "\nLong:" + lng;
double latitude = location.getLatitude();
double longitude = location.getLongitude();
Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
List<Address> addresses = gc.getFromLocation(latitude, longitude, 1);
StringBuilder sb = new StringBuilder();
if (addresses.size() > 0) {
Address address = addresses.get(0);
```

```
for (int i = 0; i < address.getMaxAddressLineIndex(); i++)
sb.append(address.getAddressLine(i)).append("\n");
sb.append(address.getLocality()).append("\n");
sb.append(address.getPostalCode()).append("\n");
sb.append(address.getCountryName());
}
addressString = sb.toString();
} catch (IOException e) {}
} else {
latLongString = "No location found";
}
myLocationText.setText("Your Current Position is:\n" +
latLongString + "\n" + addressString);
}
```

If you run the example now, it should appear as shown in Figure 7-4.

Figure 7-4