Part 3 - Location Services

Using a single API, Android provides access to various location technologies such as cellular triangulation, Wi-Fi location, and GPS. The details of each location technology are abstracted through location providers, allowing applications to obtain a location in the same way regardless of the provider that is used. The process for obtaining a location involves:

- Setting the required permissions for accessing location.
- Getting a reference to the LocationManager class.
- Using the LocationManager to request location updates for a specified provider.
- Handling a callback when the location changes.

Permissions

You have already seen location services at work earlier when the MyLocationOverlay class was used to display the current location on a map. This procedure used internal location services and, as such, it was necessary to set at least one required permission for accessing location. The permissions related to location are:

- ACCESS_FINE_LOCATION – Allows an application access to GPS.
- ACCESS_COARSE_LOCATION – Allows an application access to cellular and Wi-Fi location.

The permissions we need to set depend upon the requirements of the application. GPS will give the most accurate location, use the most power, and will only work outdoors. Cellular will use the least power and give the least accurate location. Using Wi-Fi as a location service will yield results that fall between GPS and cellular in both accuracy and power. Also, cellular and Wi-Fi will both work indoors.

Setting either, or both, of these permissions tells Android that it needs access to the location provider for the given location technology.

The Location Manager

Access to the location service happens through the LocationManager class. An application can get a reference to the LocationManager by calling GetSystemService as shown below:

```
LocationManager _locMgr;
â¦
_locMgr = GetSystemService (Context.LocationService) as LocationManager;
```

The LocationManager is kept as a class variable here because it will need to be called at various points in the Activity lifecycle.

Requesting Location Updates

For example, the RequestLocationUpdates and RemoveUpdates methods are called from the

OnResume and OnPause methods respectively, as shown below:

```
protected override void OnResume ()
{
	base.OnResume ();

	_locMgr.RequestLocationUpdates (
		LocationManager.GpsProvider, 2000, 1, this);
}

protected override void OnPause ()
{
	base.OnPause ();
	_locMgr.RemoveUpdates (this);
}
```

The RequestionLocationUpdates method tells the location services that an application that is using a specified provider would like to start receiving location updates. It also allows you to specify time and distance thresholds; these will let you control update frequency.

The RemoveUpdates method tells the location service to stop sending updates. By calling this in OnPause, you are able to conserve power if an application doesn't need location updates while its Activity is not on the screen.

Location Providers

The code above requests location from the GPS provider. If the ACCESS_FINE_LOCATION permission is set, the application will have access to the GPS provider. However, GPS may not be available for other reasons, such as if the device is indoors or does not have a GPS receiver.

If you set ACCESS_COARSE_LOCATION as well, you can use the GetBestProvider method to ask for the best available location provider. The GetBestProvider method will then use a Criteria object to set the location requirements for the provider, such as accuracy and power. The following code shows how to get the best available provider and use it when requesting location updates:

```
var locationCriteria = new Criteria();

locationCriteria.Accuracy = Accuracy.NoRequirement;
locationCriteria.PowerRequirement = Power.NoRequirement;

string locationProvider = _locMgr.GetBestProvider(locationCriteria, true);

_locMgr.RequestLocationUpdates (locationProvider, 2000, 1, this);
```

Power Considerations

The provider returned by the call to GetBestProvider will return a location provider that will use internal device hardware to get location updates. There are three types of technologies that are used internally. The technology that is used depends upon the type of location provider, as listed below:

- **Cellular Location** – Uses the least battery and is the least accurate.
- **Wi-Fi Location** – More accurate than cellular, but uses more battery.
- **GPS Location** – Most accurate and uses the most battery.

As mentioned earlier in the permission section, the finer grained the location data, the more power will be used and consequently the faster the battery will be depleted.

In addition to starting and stopping location updates in the OnResume and OnPause methods of the Activity, tuning the update interval can also conserve battery. Use arguments in the RequestLocationUpdates method to accomplish this. For example, the code listed above requested location updates every 2000ms and only when the location changed more than 1m.

```
_locMgr.RequestLocationUpdates (locationProvider, 2000, 1, this);
```

By increasing these thresholds, location services would provide updates less often. This would conserve battery life.

Receiving Location Updates

Once the LocationManager has requested location updates, the application can receive callbacks by implementing the ILocationListener interface. In particular, implement the OnLocationChanged method to receive location updates.

For example, you can implement ILocationListener in the Activity class and update the UI with the new location information by simply writing the latitude and longitude to a TextView, as shown here:

```
public void OnLocationChanged (Location location)
{
	var locationText =
		FindViewById<TextView> (Resource.Id.locationTextView);

	locationText.Text = String.Format ("Latitude = {0}, Longitude = {1}",
		location.Latitude, location.Longitude);
}
```

Geocoding

Android has a Geocoder class that can convert a street address to a latitude and longitude. This is known as geocoding. It can also reverse this process, converting a latitude and longitude or location name into a street address. This is called reverse geocoding.

Consider the following example, which uses the Geocoder class to reverse geocode the location retrieved above in ILocationListener. After creating an instance of a Geocoder, the GetFromLocation method is called and it retrieves a list of addresses:

```
new Thread (new ThreadStart (() => {
	var geocdr = new Geocoder (this);
	var addresses = geocdr.GetFromLocation (location.Latitude,
		location.Longitude, 5);

	RunOnUiThread (() => {
		var addrText =
			FindViewById<TextView> (Resource.Id.addressTextView);

		addresses.ToList ().ForEach ((addr) => {
			addrText.Append (addr.ToString () + "\r\n");
		});
	});

})).Start ();
```

Note that the Geocoder requests information over the network, so it should be called on a separate

thread to avoid blocking the main thread. Each address is an instance of an Address class, providing information about the address, such as the locality, postal code, and country name. The Geocoder will attempt to find the nearest address to the location specified, returning the number of nearby addresses passed into the third argument to GetFromLocation.

To retrieve addresses based upon a location name rather than a latitude and longitude, use the GetFromLocationName method as shown in the line below:

```
var addresses = geocdr.GetFromLocationName("Harvard University", 5);
```

The full documentation for the Address class is available in the Xamarin.Android class reference .

In the previous example, ToString was called on each address, writing the results to a TextView, as shown below:


4:14
LocationActivity
Latitude = 41.8792286, Longitude = -72.5724353
Address[addressLines=[0:"305 Rye St",1:"Broad Brook, CT 06016",2:"USA"], feature=305,admin=Connecticut,sub-admin=null, locality=null,thoroughfare=Rye St, postalCode=06016,countryCode=US, countryName=United States,hasLatitude=true, latitude=41.879073,hasLongitude=true, longitude=-72.57316,phone=null,url=null, extras=null]
Address[addressLines=[0:"Melrose, CT 06016",1:"USA"], feature=06016,admin=Connecticut,sub-admin=null,locality=null,thoroughfare=null, postalCode=06016,countryCode=US, countryName=United States,hasLatitude=true, latitude=41.9142515,hasLongitude=true, longitude=-72.5323139,phone=null,url=null, extras=null]
Address[addressLines=[0:"East Windsor, CT",1:"USA"],feature=East Windsor, admin=Connecticut,sub-admin=Hartford, locality=East Windsor,thoroughfare=null, postalCode=null,countryCode=US, countryName=United States,hasLatitude=true, latitude=41.9161361,hasLongitude=true, longitude=-72.5578799,phone=null,url=null,

Source URL: http://docs.xamarin.com/guides/android/platform_features/maps_and_location/part_3_-_location_services