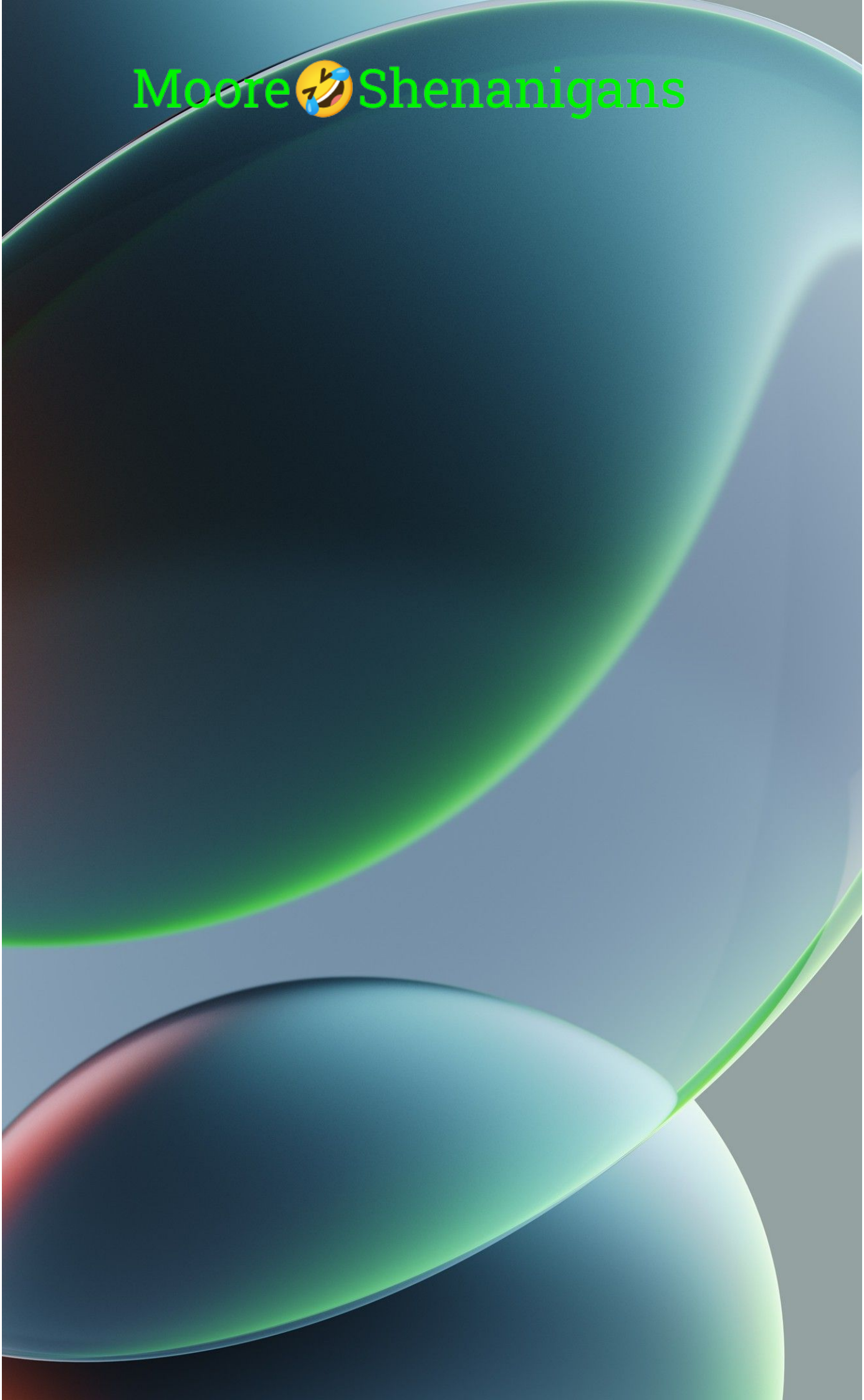
Moore🤣Shenanigans

Android Telephony

The telephony APIs let your applications access the underlying telephone hardware, making it possible to create your own dialer — or integrate call handling and phone state monitoring into your applications.

Due to security concerns, the current Android SDK does not allow you to create your own "in call" application — the screen that is displayed when an incoming call is received or an outgoing call has been initiated.

Rather than creating a new dialer implementation, the following sections focus on how to monitor and control phone, service, and cell events in your applications to augment and manage the native phonehandling functionality.

Making Phone Calls

The best practice is to use Intents to launch a dialer application to initiate new phone calls. There are two Intent actions you can use to dial a number; in both cases, you should specify the number to dial using the tel: schema as the data component of the Intent:

- Intent.ACTION_CALL Automatically initiates the call, displaying the in-call application. You should only use this action if you are replacing the native dialer, otherwise you should use the ACTION_DIAL action as described below. Your application must have the CALL_PHONE permission granted to broadcast this action.
- Intent.ACTION_DIAL Rather than dial the number immediately, this action starts a dialer application, passing in the specifi ed number but allowing the dialer application to manage the call initialization (the default dialer asks the user to explicitly initiate the call). This action doesn't require any permissions and is the standard way applications should initiate calls.

The following skeleton code shows the basic technique for dialing a number:

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:1234567"));
startActivity(intent);
```

By using Intents to announce your intention to dial a number, your application can remain fully decoupled from a particular hardware implementation used to initiate the call. For example, should you replace the existing dialer with a hybrid that allows IP-based telephony, using Intents to dial a number from other applications will let you leverage that new functionality without needing to change each application.

Monitoring Phone State and Phone Activity

The Android telephony API lets you monitor phone state, retrieve incoming phone numbers, and manage calls. Access to the telephony APIs is managed by the Telephony Manager, accessible using the getSystemService method as shown below:

```
String srvcName = Context.TELEPHONY_SERVICE;
TelephonyManager telephonyManager =
(TelephonyManager)getSystemService(srvcName);
```

In order to monitor and manage phone state, your application will need a READ_PHONE_STATE permission, as shown in the following XML code snippet:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Changes to the phone state are announced to other components using the PhoneStateListener class. Extend the PhoneStateListener to listen for, and respond to, phone state change events including call state (ringing, off hook, etc.), cell location changes, voice-mail and call-forwarding status, phone service changes, and changes in mobile signal strength.

To react to phone state change events, create a new Phone State Listener implementation, and override the event handlers of the events you want to react to. Each handler receives parameters that indicate the new phone state, such as the current cell location, call state, or signal strength.

The following code highlights the available state change handlers in a skeleton Phone State Listener implementation:

```
PhoneStateListener phoneStateListener = new PhoneStateListener() {
public void onCallForwardingIndicatorChanged(boolean cfi) {}
public void onCallStateChanged(int state, String incomingNumber) {}
```

```
public void onCellLocationChanged(CellLocation location) {}
public void onDataActivity(int direction) {}
public void onDataConnectionStateChanged(int state) {}
public void onMessageWaitingIndicatorChanged(boolean mwi) {}
public void onServiceStateChanged(ServiceState serviceState) {}
public void onSignalStrengthChanged(int asu) {}
};
```

Once you've created your own Phone State Listener, register it with the Telephony Manager using a bitmask to indicate the events you want to listen for, as shown in the following code snippet:

```
telephonyManager.listen(phoneStateListener,
PhoneStateListener.LISTEN_CALL_FORWARDING_INDICATOR |
PhoneStateListener.LISTEN_CALL_STATE |
PhoneStateListener.LISTEN_CELL_LOCATION |
PhoneStateListener.LISTEN_DATA_ACTIVITY |
PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |
PhoneStateListener.LISTEN_MESSAGE_WAITING_INDICATOR |
PhoneStateListener.LISTEN_SERVICE_STATE |
PhoneStateListener.LISTEN_SIGNAL_STRENGTH);
```

To unregister a listener, call listen and pass in PhoneStateListener.LISTEN_NONE as the bit fi eld parameter, as shown below:

```
telephonyManager.listen(phoneStateListener, PhoneStateListener.LISTEN_NONE);
```

In the following sections, you'll learn how to use the Phone State Listener to monitor incoming calls, track cell location changes, and monitor service changes.

Monitoring Phone Calls

One of the most popular reasons for monitoring phone state changes is to detect, and react to, incoming and outgoing phone calls.

Calls can be detected through changes in the phone's call state. Override the onCallStateChanged method in a Phone State Listener implementation, and register it as shown below to receive notifi cations when the call state changes:

```
PhoneStateListener callStateListener = new PhoneStateListener() {
public void onCallStateChanged(int state, String incomingNumber) {
// TODO React to incoming call.
}
};

telephonyManager.listen(callStateListener,
PhoneStateListener.LISTEN_CALL_STATE);
```

The onCallStateChanged handler receives the phone number associated with incoming calls, and the state parameter represents the current call state as one of the following three values:

- ❑ TelephonyManager.CALL_STATE_IDLE When the phone is neither ringing nor in a call
- ❑ TelephonyManager.CALL_STATE_RINGING When the phone is ringing
- ❑ TelephonyManager.CALL_STATE_OFFHOOK If the phone is currently on a call

Tracking Cell Location Changes

You can get notifi cations whenever the current cell location changes by overriding onCellLocationChanged on a Phone State Listener implementation. Before you can register to listen for cell location changes, you need to add the ACCESS_COARSE_LOCATION permission to your application manifest.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

The onCellLocationChanged handler receives a CellLocation object that includes methods for extracting the cell ID (getCid) and the current LAC (getLac).

The following code snippet shows how to implement a Phone State Listener to monitor cell location changes, displaying a Toast that includes the new location's cell ID:

```
PhoneStateListener cellLocationListener = new PhoneStateListener() {
public void onCellLocationChanged(CellLocation location) {
GsmCellLocation gsmLocation = (GsmCellLocation)location;
Toast.makeText(getApplicationContext(),
String.valueOf(gsmLocation.getCid()),
Toast.LENGTH_LONG).show();
}
};
telephonyManager.listen(cellLocationListener,
PhoneStateListener.LISTEN_CELL_LOCATION);
```

Tracking Service Changes

The onServiceStateChanged handler tracks the service details for the device's cell service. Use the ServiceState parameter to find details of the current service state.

The getState method on the Service State object returns the current service state as one of:

- ServiceState.STATE_IN_SERVICE Normal phone service is available.
- ServiceState.STATE_EMERGENCY_ONLY Phone service is available but only for emergency calls.
- ServiceState.STATE_OUT_OF_SERVICE No cell phone service is currently available.
- ServiceState.STATE_POWER_OFF The phone radio is turned off (usually when airplanemode is enabled).

A series of getOperator* methods is available to retrieve details on the operator supplying the cell phone service, while getRoaming tells you if the device is currently using a roaming profile.

The following example shows how to register for service state changes and displays a Toast showing the operator name of the current phone service:

```
PhoneStateListener serviceStateListener = new PhoneStateListener() {
public void onServiceStateChanged(ServiceState serviceState) {
if (serviceState.getState() == ServiceState.STATE_IN_SERVICE) {
String toastText = serviceState.getOperatorAlphaLong();
Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_SHORT);
}
}
};
telephonyManager.listen(serviceStateListener,
PhoneStateListener.LISTEN_SERVICE_STATE);
```