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.

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.

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.

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 specified 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);
```

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 onCallState Changed method in a Phone State Listener implementation, and register it as shown below to receive notifications 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:

a) TelephonyManager.CALL_STATE_IDLE When the phone is neither ringing nor in a call

b) TelephonyManager.CALL_STATE_RINGING When the phone is ringing

c) TelephonyManager.CALL_STATE_OFFHOOK If the phone is currently on a call

- (void)gestureChanged:(struct _GSEvent)event {

CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);

TRACKING CELL LOCATION CHANGES

We can get notifications 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 airplane mode 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);
```