11. SDK Components

Leveraging the Rich Android SDK

SDK Components

Access hardware features using built-in Android libraries and the Google Play SDK

- Taking a photo/video with the camera.
- Accessing and playing photo/video/audio.
- Accessing hardware sensors like accelerometers, light sensors, etc.
- Using the Locations API.
- Using the Maps API.

Camera

The camera implementation depends on the **level of customization** required.

- **The easy way** - launch the camera with an intent, designating a file path, and handle the onActivityResult.
- **The hard way** - use the Camera API to embed the camera preview within your app, adding your own custom controls.

Camera via Intents

```
public void onLaunchCamera(View view) {
    File mediaStorageDir = new File(
        Environment.getExternalStoragePublicDirectory(Environment.
DIRECTORY_PICTURES),
        "MyCameraApp");

    // Create the storage directory if it does not exist
    if (!mediaStorageDir.exists() && !mediaStorageDir.mkdirs()){
        Log.d("MyCameraApp", "failed to create directory");
    }
    fileUri = Uri.fromFile(new File(mediaStorageDir.getPath() + File.separator +
                "photo.jpg"));

    // create Intent to take a picture and return control to the calling application
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the image file name

    // start the image capture Intent
    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

Camera via Camera API

Integrating the camera via the Camera API involves several steps:

1. Detecting and accessing the camera
2. Creating a preview class by extending SurfaceView
3. Building a custom layout including the preview view
4. Setup capture listeners
5. Capture and save files
6. Release the camera

Accessing Media

Similar to the camera, the media picker implementation depends on the **level of customization** required.

- **The easy way** - launch the Gallery with an intent, and get the media URI in onActivityResult.
- **The hard way** - fetch thumbnail and full-size URIs from the MediaStore ContentProvider.

Accessing Media via Intents

This will launch the Camera app on the phone. When the user finishes taking the photo, it will return to your app.

```
public void onPickPhoto(View view) {
    Intent intent = new Intent(Intent.ACTION_PICK,
        android.provider.MediaStore
        .Images.Media.EXTERNAL_CONTENT_URI);
    startActivityForResult(intent, 0);
}
```

Accessing Media via MediaStore

```
public ArrayList<String> getMediaPaths {
    String[] projection = {MediaStore.Images.Media.DATA};
    int columnIndex =
        cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);

    cursor = managedQuery(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        projection, // Which columns to return
        null,       // Return all rows
        null,       // selection criteria args
        null);      // sort order

    ArrayList<String> paths = new ArrayList<String>();
    while (cursor.moveToNext()) {
        paths.add(cursor.getString(columnIndex));
    }
}
```

Video Player

VideoView is the default wrapper for MediaPlayer for playing videos.

```
public void playUrl(String url) {
    videoView.setVideoPath(url);
    videoView.setMediaController(new MediaController(this));
    videoView.requestFocus();
    videoView.start();
}
```

Hardware Sensors

Different devices have a variety of sensors that can be accessed via the Sensor framework. Possible tasks include:

- List available sensors
- Determine sensor capabilities (range, resolution, etc)
- Acquire raw sensor data
- Register sensor event listeners

Listing Hardware Sensors

List available hardware sensors:

```
SensorManager sensorManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);

List<Sensor> deviceSensors =
    sensorManager.getSensorList(Sensor.TYPE_ALL);
for (Sensor s : deviceSensors) {
    Log.d("DEBUG", s.getName());
}
```

Listening to Sensors

Register for Sensor events, unregister onPause.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager =
        (SensorManager)getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    mSensorManager.registerListener(this, mLight,
        SensorManager.SENSOR_DELAY_NORMAL);
}

public void onSensorChanged(SensorEvent event) {
    ...
}
```

Google Play SDK

Google APIs such as Maps, Locations, In-App Purchasing, Licensing are provided in the Google Play SDK

- Install the Google Play Services SDK
- Import the library from: `<android-sdk>/extras/google/google_play_services/libproject/google-play-services_lib/`
- Setup your project with the library

Location API

The Location API is a higher-level API that wraps the underlying location sensor. You can accomplish tasks like:

- Connect to the location sensor
- Register for updates or accuracy changes
- Get last location
- Register for sensor connection events

Location API

Connect to LocationClient:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mLocationClient = new LocationClient(this, this, this);
    mLocationClient.connect();
}

public void onConnected(Bundle arg0) {
    Location mCurrentLocation = mLocationClient.getLastLocation();
    Log.d("DEBUG", "current location: " + mCurrentLocation.toString());
}
```

Location API

Register for location updates

```
public void onLocationChanged(Location location) {
    // Report to the UI that the location was updated
    String msg = "Updated Location: " +
        Double.toString(location.getLatitude()) + "," +
        Double.toString(location.getLongitude());
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
}
```

Maps API

Access the Google Maps API by following these steps:

1. Obtain a Maps API key
   https://developers.google.com/maps/documentation/android/start#the_google_maps_api_key
2. Add the key to your app in the AndroidManifest.xml

```
<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="API_KEY"/>
```

3. Specify permissions
   https://developers.google.com/maps/documentation/android/start#specifying_permissions
4. Embed SupportMapFragment

Maps API

SupportMapFragment contains an underlying GoogleMap that has rich functionality such as:

1. Animated camera control, zooming to various longitudes and latitudes
2. Dropping markers and custom drawing
3. Map gestures such as zoom, pan, tilt, rotate
4. Listening to map click events

Maps API - example

Sample code for zooming the map to the current location.

```
Location mCurrentLocation = mLocationClient.getLastLocation();
LatLng currentLatLng = new
    LatLng(mCurrentLocation.getLatitude(),
           mCurrentLocation.getLongitude());
SupportMapFragment fragment = (SupportMapFragment)
    getSupportFragmentManager().findFragmentById(R.id.map);

GoogleMap map = fragment.getMap();
map.moveCamera(CameraUpdateFactory.newLatLngZoom
(currentLatLng, 15));
```

Components Wrap-up

- **Camera** can be used via Intents or Camera API
- **MediaStore** can be used to access Media
- **VideoView** can be used to play video
- **Hardware sensors** can be listened in on
- **Location API** can be used to detect position'
- **Maps API** can be used to display Google Maps