CHAPTER 1

안드로이드의 이미지

이 장에서는 안드로이드의 이미지 캡처와 저장에 관한 기본 내용을 살펴본다. 우선 안드로이드에 내장되어 제공되는 기능부터 접해보고, 다른 소프트웨어로 이어나갈 것이다. 이미지를 캡처하고 저장하는 데 사용할 내장 기능은 안드로이드의 미디어 기능을 전반적으로 소개해주는 훌륭한 첫걸음이 된다. 그리고 이 책 후반부에서 오디오와 비디오를 다룰 때 든든한 기초 지식이 돼줄 것이다.

따라서 이제부터 내장 Camera 애플리케이션을 시작으로 미디어스토어(MediaStore)를 활용한 내장 미디어 및 메타데이터 저장 메커니즘까지 심도 있게 진행할 것이다. 디지털 제품이나 이미지 프로세싱 소프트웨어 분야에서 메타데이터를 공유하기 위한 표준인 EXIF를 효율적으로 활용하는 방법과 메모리를 적게 사용하는 방법도 살펴볼 것이다.

내장 Camera 애플리케이션을 사용하여 이미지 캡처하기

스마트폰은 컴퓨터를 빠르게 대신해 가고 있으며, 많은 면에서 다양한 디지털 제품의 자

리를 이미 꿰찼다고 할 수 있다. 스마트폰에 음성 통화와 상관없는 장치로 가장 먼저 장착된 하드웨어를 말해보라고 하면, 단연 카메라가 으뜸일 것이다. 지금은 카메라가 달려 있지 않은 스마트폰을 사는 것 자체가 스트레스라고 해도 틀린 말이 아니다. 물론 안드로이드 기반의 스마트폰도 예외는 아니어서 안드로이드 SDK는 애초에 출시될 때부터 이미지를 캡처하기 위해 내장 카메라에 액세스할 수 있었다.

안드로이드에서 가장 손쉽게, 그리고 직관적으로 작업을 처리하는 방법은 인텐트(intent)를 통해 기기에 이미 존재하는 소프트웨어를 활용하는 것이다. 인텐트란 안드로이드의 핵심 구성요소로서, 안드로이드 문서에는 "수행될 액션을 설명한 것"으로 기술되어 있다. 실제로 인텐트는 다른 애플리케이션이 어떤 일을 하도록 트리거하거나 단일 애플리케이션에서 액티비티 사이를 왔다 갔다 하는 데 사용된다.

적절한 하드웨어, 즉 카메라를 갖춘 안드로이드 기기라면 전부 다 Camera 애플리케이션을 제공한다. Camera 애플리케이션에는 인텐트 필터가 포함되어 있는데, 이를 사용하면 개발자들은 캡처 루틴을 직접 작성하지 않고도 Camera 애플리케이션과 동일한 이미지 캡처 기능을 제공할 수 있다.

인텐트 필터는 어떤 애플리케이션을 작성하는 프로그래머가 그 애플리케이션에서 어떤 기능을 제공하겠다고 지정하기 위한 수단이다. 어떤 애플리케이션의 Android Manifest.xml 파일에 인텐트 필터를 지정하면 안드로이드는 그 애플리케이션이, 구체적으로는 그 인텐트 필터를 담은 액티비티가 어떤 태스크를 명령에 따라 수행하는 것으로 이해한다.

Camera 애플리케이션에는 다음과 같이 매니페스트 파일에 지정된 인텐트 필터가 제공된다. 여기에 나타낸 인텐트 필터는 "Camera" 액티비티 태그 안에 들어간다.

```
<intent-filter>
    <action android:name="android.media.action.IMAGE_CAPTURE" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

인텐트를 통해 Camera 애플리케이션을 활용하기 위해서는 이 필터가 잡게 될 인텐트를 다음과 같이 구성해야 한다.

```
Intent i = new Intent("android.media.action.IMAGE_CAPTURE");
```

실제로는 이 액션 문자열로 직접 인텐트를 생성하려고 하지 않을 것 같다. 그 대신 MediaStore 클래스에 ACTION_IMAGE_CAPTURE라는 상수가 지정돼 있는데, 문자열보다 이 상수를 사용해야 한다. 문자열이 바뀌기라도 하면 덩달아 상수도 바뀔 수 있기 때문이다. 따라서 상수를 사용하면 나중에 일어날지도 모르는 일을 미연에 막으면서 호출할 수 있다.

```
Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
startActivity(i);
```

기본 안드로이드 액티비티에 이 인텐트를 사용하면 디폴트 Camera 애플리케이션이 그림 1-1과 같이 이미지 모드로 시작된다.

:: **그림 1-1** 인텐트로 호출된 내장 Camera 애플리케이션이 에뮬레이터에서 실행된 모습

Camera 앱에서 데이터 리턴하기

내장 Camera 애플리케이션으로 이미지를 캡처하는 것이 쓸모가 있으려면 이미지가 캡

처될 때 호출 액티비티한테 해당 이미지를 Camera 애플리케이션이 리턴하도록 해야 한다. 그러려면 우리의 액티비티에서 startActivity 메소드를 startActivityForResult 메소드로 교체한다. startActivityForResult 메소드를 사용해야 Camera 애플리케이션에서 리턴해준 데이터에 접근할 수 있다. 리턴된 이 데이터는 사용자가 비트맵으로 캡처한 이미지다.

다음은 기본 예제다.

```
package com.apress.proandroidmedia.ch1.cameraintent;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.widget.ImageView;

public class CameraIntent extends Activity {

    final static int CAMERA_RESULT = 0;

    ImageView imv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(i, CAMERA_RESULT);
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);

        if (resultCode == RESULT_OK)
        {
            Bundle extras = intent.getExtras();
            Bitmap bmp = (Bitmap) extras.get("data");
```

```
            imv = (ImageView) findViewById(R.id.ReturnedImageView);
            imv.setImageBitmap(bmp);
        }
    }
}
```

프로젝트의 layout/main.xml 파일에 다음과 같은 내용이 들어 있어야 한다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <ImageView android:id="@+id/ReturnedImageView" android:layout_width="wrap_content" ↵
 android:layout_height="wrap_content"></ImageView>
</LinearLayout>
```

다음은 이 연습을 마무리하기 위한 AndroidManifest.xml의 내용이다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0" package="com.apress.proandroidmedia.ch1.cameraintent">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".CameraIntent"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

이미지는 인텐트를 통해 넘어온 extra(다용도 가방)에 담겨 Camera 애플리케이션에서 리턴된다. 이 인텐트는 onActivityResult 메소드의 호출 액티비티한테 전송된다. 그리고 이 extra의 이름은 'data' 이고, Bitmap 객체를 담는다. 이 객체는 제네릭(generic) 객체에서 타입이 변환되어야 한다.

```
// 인텐트에서 extra를 가져온다.
Bundle extras = intent.getExtras();

// 리턴된 이미지를 extra에서 가져온다.
Bitmap bmp = (Bitmap) extras.get("data");
```

레이아웃 XML 파일(layout/main.xml)에는 ImageView가 있다. ImageView는 제네릭 View의 확장판으로, 이미지 표시를 지원한다. ImageView에서 id는 ReturnedImageView로 지정되었으므로 우리의 액티비티에서는 이 객체를 가리키는 참조가 필요하고, 이렇게 하면 캡처한 이미지를 사용자가 볼 수 있다.

ImageView 객체를 가리키는 참조를 얻기 위해 Activity 클래스에 지정된 표준 findViewById 메소드를 사용한다. 그래야 레이아웃 XML 파일에 지정된 요소들이 setContentView를 통해 프로그램상으로 참조될 수 있는데, 해당 요소의 id를 넘겨주면 된다. 이 예제에서 ImageView 객체는 다음과 같이 XML 파일에 지정된다.

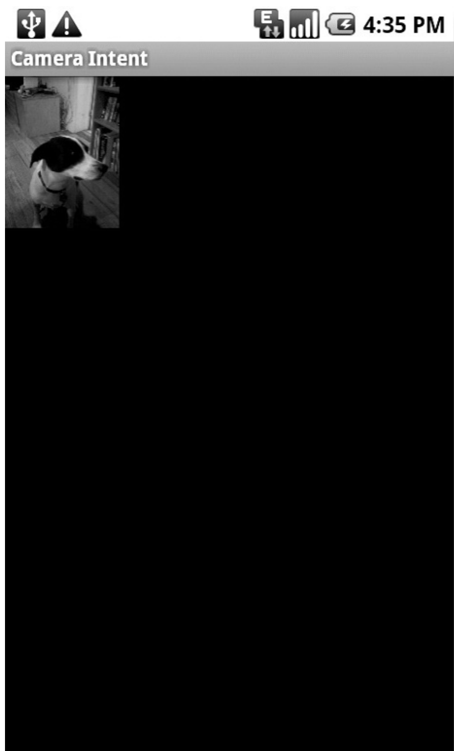
```
<ImageView android:id="@+id/ReturnedImageView" android:layout_width="wrap_content" ↵
android:layout_height="wrap_content"></ImageView>
```

ImageView를 참조하여 Camera 애플리케이션에서 가져온 Bitmap을 표시하게 하려면 다음 코드를 작성한다.

```
imv = (ImageView) findViewById(R.id.ReturnedImageView);imv.setImageBitmap(bmp);
```

이 예제를 실행하면 결과 이미지가 작게 나타난다. (필자의 스마트폰에서는 가로가 121픽셀, 세로가 162픽셀이었다. 기기마다 디폴트값이 다를 수 있다.) 이는 버그가 아니라 원래 그런 것이다. Camera 애플리케이션은 인텐트로 트리거되면 풀 사이즈 이미지를 호출 액

티비티에 리턴하지 않는다. 풀 사이즈 이미지를 리턴하면 메모리를 많이 잡아먹으므로 모바일 기기에서는 대개 그 특성상 어느 정도 제한을 둔다. 그런 이유로 Camera 애플리케이션은 리턴된 인텐트의 작은 썸네일 이미지를 리턴한다. 결과 화면은 그림 1-2다.



:: **그림 1-2** ImageView에 표시된 121x162픽셀의 이미지

큰 이미지 캡처하기

안드로이드 1.5부터 대부분의 기기에서는 Camera 애플리케이션을 트리거하는 데 사용되는 인텐트한테 extra를 넘겨줄 수 있게 하여 크기 제한을 해결했다. 이 extra의 이름은 MediaStore 클래스에 EXTRA_OUTPUT이라는 상수로 지정되어 있다. 그리고 그 값(extra는 이름-값 쌍의 형태를 띤다)은 Camera 애플리케이션한테 URI 형태로 캡처 이미지가 어느 곳에 저장되는지 가리킨다.

다음 코드에서는 Camera 애플리케이션한테 이미지가 기기에 장착된 SD 카드에 저장되어야 한다고 알린다. 파일명은 myfavoritepicture.jpg다.

```
String imageFilePath = Environment.getExternalStorageDirectory().getAbsolutePath() ➥
        + "/myfavoritepicture.jpg";
File imageFile = new File(imageFilePath);
Uri imageFileUri = Uri.fromFile(imageFile);

Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
startActivityForResult(i, CAMERA_RESULT);
```

Note 이미지 파일의 URI를 생성하기 위한 이 코드는 다음과 같이 줄일 수 있다.

```
imageFileUri = Uri.parse("file:///sdcard/myfavoritepicture.jpg");
```

이렇게 메소드를 사용하면 단순히 코드를 줄이는 차원에서 끝나지 않는다. 우선 기기 종속성을 낮출 수 있고, 로컬 파일시스템의 URI 문법이나 SD 카드의 이름 지정 규칙이 변경될지도 모르므로 더욱더 확실하게 나중을 대비하는 효과까지 누릴 수 있다.

큰 이미지 표시하기

이미지를 로드하고 표시하려면 메모리가 많이 필요하다. 예를 들어, HTC G1(최초의 안드로이드폰)에는 320만 화소짜리 카메라가 달렸다. 이 정도 성능이면 대개 2048x1536픽셀의 이미지를 캡처한다. 이 크기의 32비트 이미지를 표시하려면 메모리가 대략 13MB 이상 필요한데, 이 만큼이면 우리의 애플리케이션에서 메모리가 부족할 것이라고 분명하게는 말할 수 없지만, 그럴 가능성이 다분하다.

안드로이드는 BitmapFactory라는 유틸리티 클래스를 제공하는데, 이 클래스에는 다양한 소스에서 비트맵 이미지를 로드하는 일련의 정적 메소드가 제공된다. 여기서는 파일에서 이미지를 로드하여 우리의 원래 액티비티에서 표시할 것이다. 다행히도 BitmapFactory의 메소드들은 BitmapFactory.Options 클래스에서 지원된다. 이 클래스를 사용하여 Bitmap이 메모리에 읽히는 방식을 정의할 수 있다. 구체적으로 말하면 이미지가 로드될

때 BitmapFactory가 사용하는 샘플 크기를 정할 수 있다. BitmapFactory.Options의 inSampleSize 파라미터는 결과 이미지인 Bitmap이 로드됐을 때보다 얼마만큼 작아지는지 그 비율을 나타낸다. 가령, 다음 코드에서처럼 inSampleSize를 8로 지정하면 원래 이미지 크기의 8분의 1로 작아진 이미지가 생성된다.

```
BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
bmpFactoryOptions.inSampleSize = 8;
Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);
imv.setImageBitmap(bmp);
```

이렇게 하면 큰 이미지를 손쉽게 로드할 수 있지만, 이미지의 원래 크기도 표시될 화면의 크기도 고려하지 못하게 된다. 화면에 근사하게 딱 맞도록 이미지를 작게 한다면 참 좋을 텐데 말이다.

아래 이어지는 코드에서는 어떻게 하면 이미지를 로드할 때 거쳐야 하는 축소 샘플링의 정도를 화면의 크기에 맞춰 결정할 수 있는지 보여준다. 아래 방법을 사용하면 이미지는 화면에서 최대 크기로 채워진다. 하지만 이미지가 가로든 세로든 어느 하나가 100픽셀로만 표시된다면 화면 크기 대신 이 수치가 대신 사용된다. 화면 크기는 다음 코드로 얻을 수 있다.

```
Display currentDisplay = getWindowManager().getDefaultDisplay();
int dw = currentDisplay.getWidth();
int dh = currentDisplay.getHeight();
```

이미지의 치수를 정하는 것은 축소 샘플링 계산에 필요한 과정인데, BitmapFactory.Options.inJustDecodeBounds 변수를 true로 설정해놓고 BitmapFactory와 BitmapFactory.Options를 사용한다. 이렇게 하면 BitmapFactory 클래스는 이미지 자체를 디코딩하려고 하지 않고 이미지의 가로 세로를 넘겨준다. 그리고 BitmapFactory.Options.outHeight 변수와 BitmapFactory.Options.outWidth 변수가 채워진다.

```
// 이미지가 아니라 이미지의 치수를 로드한다.
BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
bmpFactoryOptions.inJustDecodeBounds = true;
Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

int heightRatio = (int)Math.ceil(bmpFactoryOptions.outHeight/(float)dh);
int widthRatio = (int)Math.ceil(bmpFactoryOptions.outWidth/(float)dw);

Log.v("HEIGHTRATIO",""+heightRatio);
Log.v("WIDTHRATIO",""+widthRatio);
```

이미지의 가로(너비)와 세로(높이) 치수를 화면의 가로와 세로 치수로 단순하게 나눈 값이 비율에 해당한다. 가로와 세로 중 어느 쪽이 더 크냐에 따라 가로 비율을 선택해도 되고, 세로 비율을 선택해도 된다. BitmapFactory.Options.inSampleSize 변수로 해당 비율을 사용하면 필요한 치수에 가깝게 메모리에 로드되는 이미지를 만들어낼 수 있다. 이 경우에는 화면 자체 치수와 가깝게 된다.

```
// 이미지의 가로, 세로 중 한쪽이 화면보다 크다.
if (heightRatio > 1 && widthRatio > 1)
{
    if (heightRatio > widthRatio)
    {
        // 높이 비율이 더 커서 그에 따라 맞춘다.
        bmpFactoryOptions.inSampleSize = heightRatio;
    }
    else
    {
        // 너비 비율이 더 커서 그에 따라 맞춘다.
        bmpFactoryOptions.inSampleSize = widthRatio;
    }
}

// 실제로 디코딩한다.
bmpFactoryOptions.inJustDecodeBounds = false;
bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);
```

다음은 예제에 쓰일 전체 코드로서, 인텐트를 통해 내장 카메라를 사용하고 결과에 해당하는 이미지를 표시한다. 그림 1-3은 화면 크기에 맞춰진 이미지다.

```
package com.apress.proandroidmedia.ch1.sizedcameraintent;

import java.io.File;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.Display;
import android.widget.ImageView;

public class SizedCameraIntent extends Activity {

    final static int CAMERA_RESULT = 0;

    ImageView imv;
    String imageFilePath;

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

        imageFilePath = Environment.getExternalStorageDirectory().getAbsolutePath() +
            "/myfavoritepicture.jpg";
        File imageFile = new File(imageFilePath);
        Uri imageFileUri = Uri.fromFile(imageFile);

        Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
        startActivityForResult(i, CAMERA_RESULT);
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
```

```
        super.onActivityResult(requestCode, resultCode, intent);

        if (resultCode == RESULT_OK) {
            // ImageView의 참조를 가져온다.
            imv = (ImageView) findViewById(R.id.ReturnedImageView);

            Display currentDisplay = getWindowManager().getDefaultDisplay();
            int dw = currentDisplay.getWidth();
            int dh = currentDisplay.getHeight();

            // 이미지가 아니라 이미지의 치수를 로드한다.
            BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
            bmpFactoryOptions.inJustDecodeBounds = true;
            Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

            int heightRatio = (int) Math.ceil(bmpFactoryOptions.outHeight / (float) dh);
            int widthRatio = (int) Math.ceil(bmpFactoryOptions.outWidth / (float) dw);
            Log.v("HEIGHTRATIO", "" + heightRatio);
            Log.v("WIDTHRATIO", "" + widthRatio);

            // 두 비율 다 1보다 크면 이미지의 가로, 세로 중 한쪽이 화면보다 크다.
            if (heightRatio > 1 && widthRatio > 1) {
                if (heightRatio > widthRatio) {
                    // 높이 비율이 더 커서 그에 따라 맞춘다.
                    bmpFactoryOptions.inSampleSize = heightRatio;
                }
                else
                {
                    // 너비 비율이 더 커서 그에 따라 맞춘다.
                    bmpFactoryOptions.inSampleSize = widthRatio;
                }
            }

            // 실제로 디코딩한다.
            bmpFactoryOptions.inJustDecodeBounds = false;
            bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

            // 이미지를 표시한다.
            imv.setImageBitmap(bmp);
        }
    }
}
```

이 코드에는 다음 layout/main.xml 필요하다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <ImageView android:id="@+id/ReturnedImageView" android:layout_width="wrap_content"↵
android:layout_height="wrap_content"></ImageView>
</LinearLayout>
```

:: **그림 1-3** 화면 크기로 맞춰진 ImageView에 표시된 결과 이미지

이미지 저장과 메타데이터

안드로이드는 애플리케이션끼리 데이터를 공유할 수 있는 표준 방법을 제공한다. 이를 담당하는 클래스들이 **콘텐트 프로바이더(Content Provider)**다. 콘텐트 프로바이더는 다양한 타입의 데이터를 저장하고 검색하는 데 필요한 표준 인터페이스를 제공한다.

이미지용 표준 콘텐트 프로바이더(오디와 비디오도 함께 지원)가 MediaStore다. 기기의 표준 위치에 파일을 저장할 수 있도록 해주는 MediaStore는 저장되는 파일의 메타데이터를 저장하고 검색하는 기능 또한 제공한다. 메타데이터는 데이터에 관한 데이터다. 다시 말해 데이터에 관한 정보, 가령 그 데이터의 크기나 이름 등을 파일 자체에 가질 수 있다는 의미다. 하지만 MediaStore는 이 외에도 다양한 추가 데이터, 예를 들어 제목이나 설명, 위도와 경도까지도 담을 수 있다.

앞에서 다룬 SizedCameraIntent 액티비티를 변경하여 MediaStore를 본격적으로 활용할 텐데, 지난번처럼 이미지를 SD 카드에 임의의 파일로 저장하지 않고 저장 위치나 메타데이터를 구체적으로 다룰 것이다.

이미지의 URI 얻기

이미지 저장에 필요한 표준 위치를 얻으려면 먼저 MediaStore의 참조부터 가져온다. 그러기 위해서는 **콘텐트 리졸버(Content Resolver)**를 사용한다. 콘텐트 리졸버란 한 마디로 MediaStore라는 콘텐트 프로바이더에 접근하는 수단이다.

콘텐트 리졸버는 특정 URI를 넘겨주어 콘텐트 프로바이더인 MediaStore에 인터페이스를 제공한다. 새로운 이미지를 삽입할 때 필요한 메소드는 insert고, 그 URI는 android.provider.MediaStore.Images.Media 클래스의 EXTERNAL_CONTENT_URI라는 상수에 담긴다. 이 말인즉슨, 이미지를 기기의 주 외부 볼륨, 즉 SD 카드에 저장하겠다는 의미다. 기기의 내장 메모리에 이미지를 저장하겠다면 INTERNAL_CONTENT_URI 상수를 사용한다. 하지만 이미지나 오디오, 비디오 등이 크기가 비교적 크기 때문에 EXTERNAL_CONTENT_URI를 사용하여 미디어를 저장하는 것이 일반적이라 하겠다.

삽입(insert)을 호출하면 URI가 리턴되는데, 이 URI를 사용하여 이미지 파일의 바이너리 데이터를 쓸 수 있다. 여기서는 CameraActivity를 사용하고 있으므로 Camera 애플리케이션을 트리거하는 인텐트의 형태로 extra를 넘겨준다.

```
Uri imageFileUri = getContentResolver().insert(
    Media.EXTERNAL_CONTENT_URI, new ContentValues());

Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
startActivityForResult(i, CAMERA_RESULT);
```

이 코드에서 새로운 ContentValues 객체가 넘어가는 행을 보자. ContentValues 객체는 레코드에 연결할 메타데이터다. 위의 경우에는 빈 ContentValues 객체를 넘겨준다.

연결되는 메타데이터에 정보를 미리 채워놓기

메타데이터의 내용을 미리 채워 넣으려면 put 메소드를 사용한다. ContentValues가 데이터를 이름-값의 쌍으로 받는데, 기준이 되는 이름은 android.provider.MediaStore.Images.Media 클래스에 상수로 정의돼 있다(일부 상수의 실제 위치는 android.provider.MediaStore.MediaColumns 인터페이스이며, Media 클래스가 이 인터페이스를 구현한다).

```
// 맵에 이미지의 이름과 설명을 저장한다.
ContentValues contentValues = new ContentValues(3);
contentValues.put(Media.DISPLAY_NAME, "This is a test title");
contentValues.put(Media.DESCRIPTION, "This is a test description");
contentValues.put(Media.MIME_TYPE, "image/jpeg");

// 비트맵은 없지만 일부 값이 설정된 레코드를 추가한다.
// insert()는 이 새 레코드의 URI를 리턴한다.
Uri imageFileUri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, contentValues);
```

한 번 더 얘기하면, 이 호출로 리턴되는 것은 URI로서, 이미지의 저장 위치를 지정하기 위해서 인텐트를 통해 Camera 애플리케이션에 넘겨진다.

이 URI를 Log 명령으로 출력하면 다음과 같은 내용이 된다.

```
content://media/external/images/media/16
```

여기서 우선 눈여겨볼 것은 이 내용이 브라우저에서 사용할 수 있는 여느 URL처럼 보일 수도 있지만, 웹 페이지를 전송하는 프로토콜인 http로 시작하지 않고 content로 시작한다는 점이다. 안드로이드에서는 URI가 content로 시작하면 콘텐트 프로바이더(예, MediaStore)와 사용되는 것이라고 여긴다.

저장된 이미지 검색하기

앞에서 이미지를 저장하려고 얻은 URI는 이미지에 접근하는 수단으로도 그대로 사용할 수 있다. 파일의 전체 경로를 BitmapFactory에 넘겨주는 것이 아니라, 다음과 같이 콘텐트 리졸버를 통해 해당 이미지의 InputStream을 열어 BitmapFactory에 넘겨주면 된다.

```
Bitmap bmp = BitmapFactory.decodeStream( ↵
getContentResolver().openInputStream(imageFileUri), null, bmpFactoryOptions);
```

메타데이터 추가하기

이미지를 MediaStore에 캡처하고 나서 나중에 더 많은 메타데이터를 그 이미지에 연결할 생각이면, 우리의 컨텐트 리졸버가 제공하는 update 메소드를 사용한다. update 메소드는 앞에서 사용한 insert 메소드와 한 가지만 빼고 아주 흡사하다. update 메소드는 해당 URI를 가지고 이미지 파일에 직접 접근한다.

```
// 레코드에서 제목과 설명을 업데이트한다.
ContentValues contentValues = new ContentValues(3);
contentValues.put(Media.DISPLAY_NAME, "This is a test title");
contentValues.put(Media.DESCRIPTION, "This is a test description");
getContentResolver().update(imageFileUri,contentValues,null,null);
```

MediaStore를 이미지 저장용으로 사용하고 메타데이터를 연결할 수 있도록 CameraActivity 업데이트하기

다음은 앞선 예제의 업데이트된 내용으로서 이미지를 MediaStore에 저장한 다음, 제목이나 설명을 추가할 기회를 마련해주는 예제다. 하나 더 덧붙이면 이 버전에는 UI 요소가 몇 가지 포함됐는데, 어떤 요소가 언제 보이느냐는 애플리케이션에서 사용자가 어느 화면에 있느냐로 관리된다.

```
package com.apress.proandroidmedia.ch1.mediastorecameraintent;

import java.io.FileNotFoundException;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.provider.MediaStore.Images.Media;
import android.content.ContentValues;

public class MediaStoreCameraIntent extends Activity {

    final static int CAMERA_RESULT = 0;

    Uri imageFileUri;

    // res/layout/main.xml에 지정된 사용자 인터페이스 요소들
    ImageView returnedImageView;
    Button takePictureButton;
    Button saveDataButton;
    TextView titleTextView;
```

```
TextView descriptionTextView;
EditText titleEditText;
EditText descriptionEditText;
```

이제 두어 가지 사용자 인터페이스 요소를 집어넣을 것이다. 사용자 인터페이스 요소는 layout/main.xml에서 지정되고, 그 객체들은 앞의 코드에서 선언된다.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    // 콘텐트 뷰를 res/layout/main.xml 파일에 정의된 것으로 설정한다.
    setContentView(R.layout.main);

    // UI 요소의 참조를 가져온다.
    returnedImageView = (ImageView) findViewById(R.id.ReturnedImageView);
    takePictureButton = (Button) findViewById(R.id.TakePictureButton);
    saveDataButton = (Button) findViewById(R.id.SaveDataButton);
    titleTextView = (TextView) findViewById(R.id.TitleTextView);
    descriptionTextView = (TextView) findViewById(R.id.DescriptionTextView);
    titleEditText = (EditText) findViewById(R.id.TitleEditText);
    descriptionEditText = (EditText) findViewById(R.id.DescriptionEditText);
```

표준 액티비티인 onCreate 메소드에서는 setContentView를 호출한 뒤, 코드상으로 조절해야 하는 사용자 인터페이스 요소들을 인스턴스로 만든다. 인스턴스가 하나씩 생기면 findViewById 메소드로 가져와 적절한 타입으로 변환하는 과정을 거쳐야 한다.

```
    // 처음에는 takePictureButton만 빼고 모두 보이지 않음으로 설정한다.
    // View.GONE은 보이지 않으므로 레이아웃에서 공간을 차지하지 않는다.
    returnedImageView.setVisibility(View.GONE);
    saveDataButton.setVisibility(View.GONE);
    titleTextView.setVisibility(View.GONE);
    descriptionTextView.setVisibility(View.GONE);
    titleEditText.setVisibility(View.GONE);
    descriptionEditText.setVisibility(View.GONE);
```

계속해서 이야기를 이어나가겠다. 사용자 인터페이스 요소 전부 레이아웃 파일에서 보이지 않고 자리도 차지하지 않도록 설정했다. View.GONE은 이를 위해 setVisibility 메소드에서 사용되는 상수다. View.INVISIBLE이라는 옵션도 있는데, 이 옵션은 레이아웃에서 숨기는 것은 같지만 자리를 차지한다는 점에서 다르다.

```
        // takePictureButton이 클릭될 때
        takePictureButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v)
            {
                // 비트맵 없이 새 레코드를 추가한다.
                // 새 레코드의 URI를 리턴한다.
                imageFileUri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, ➥
new ContentValues());

                // Camera 앱을 시작한다.
                Intent i = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
                i.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
                startActivityForResult(i, CAMERA_RESULT);
            }
        });
```

takePictureButton의 OnClickListener에서 내장 카메라에 필요한 표준 인텐트를 만들고, startActivityForResult를 호출한다. onCreate 메소드를 직접 사용하지 않고 지금이 방법대로 하면 조금은 '쌈빡한' 사용자 경험을 만들 수 있다.

```
        saveDataButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v)
            {
                // MediaStore 레코드에서 제목과 설명을 업데이트한다.
                ContentValues contentValues = new ContentValues(3);
                contentValues.put(Media.DISPLAY_NAME, titleEditText.getText().toString());
                contentValues.put(Media.DESCRIPTION, descriptionEditText.getText().toString());
                getContentResolver().update(imageFileUri,contentValues,null,null);

                // 사용자에게 알린다.
                Toast bread = Toast.makeText(MediaStoreCameraIntent.this, "Record ➥
Updated", Toast.LENGTH_SHORT);
```

```
            bread.show();

            // 초기 상태로 돌아가 takePictureButton을 보이도록 설정한다.
            // 다른 UI 요소들은 숨긴다.
            takePictureButton.setVisibility(View.VISIBLE);

            returnedImageView.setVisibility(View.GONE);
            saveDataButton.setVisibility(View.GONE);
            titleTextView.setVisibility(View.GONE);
            descriptionTextView.setVisibility(View.GONE);
            titleEditText.setVisibility(View.GONE);
            descriptionEditText.setVisibility(View.GONE);
        }
    });
}
```

saveDataButton의 OnClickListener는 Camera 애플리케이션이 이미지를 리턴하고 나면 보이는데, 메타데이터를 그 이미지에 연결하는 역할을 담당한다. 구체적으로 말하면, 사용자가 EditText 요소들에 입력한 값들을 받아 ContentValues 객체를 만든다. 이 객체는 MediaStore에 든 해당 이미지의 레코드를 업데이트하는 데 사용된다.

```
protected void onActivityResult(int requestCode, int resultCode, Intent intent)
{
    super.onActivityResult(requestCode, resultCode, intent);

    if (resultCode == RESULT_OK)
    {
        // Camera 앱으로 돌아왔다.

        // takePictureButton을 숨긴다.
        takePictureButton.setVisibility(View.GONE);

        // 다른 UI 요소들을 보이게 한다.
        saveDataButton.setVisibility(View.VISIBLE);
        returnedImageView.setVisibility(View.VISIBLE);
        titleTextView.setVisibility(View.VISIBLE);
        descriptionTextView.setVisibility(View.VISIBLE);
        titleEditText.setVisibility(View.VISIBLE);
```

```
            descriptionEditText.setVisibility(View.VISIBLE);

            // 이미지 크기를 조정한다.
            int dw = 200; // 최대 200픽셀 너비
            int dh = 200; // 최대 200픽셀 높이

            try
            {
                // 이미지가 아니라 이미지의 치수를 로드한다.
                BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
                bmpFactoryOptions.inJustDecodeBounds = true;
                Bitmap bmp = BitmapFactory.decodeStream(getContentResolver(). ↵
openInputStream(imageFileUri), null, bmpFactoryOptions);

                int heightRatio = (int)Math.ceil(bmpFactoryOptions.outHeight/(float)dh);
                int widthRatio = (int)Math.ceil(bmpFactoryOptions.outWidth/(float)dw);

                Log.v("HEIGHTRATIO",""+heightRatio);
                Log.v("WIDTHRATIO",""+widthRatio);

                // 두 비율 다 1보다 크면 이미지의 가로, 세로 중 한쪽이 화면보다 크다.
                if (heightRatio > 1 && widthRatio > 1)
                {
                     if (heightRatio > widthRatio)
                     {
                          // 높이 비율이 더 커서 그에 따라 맞춘다.
                          bmpFactoryOptions.inSampleSize = heightRatio;
                     }
                     else
                     {
                          // 너비 비율이 더 커서 그에 따라 맞춘다.
                          bmpFactoryOptions.inSampleSize = widthRatio;
                     }
                }
                // 실제로 디코딩한다.
                bmpFactoryOptions.inJustDecodeBounds = false;
                bmp = BitmapFactory.decodeStream(getContentResolver(). ↵
openInputStream(imageFileUri), null, bmpFactoryOptions);

                // 이미지를 표시한다.
                returnedImageView.setImageBitmap(bmp);
```

```
            }
            catch (FileNotFoundException e)
            {
                Log.v("ERROR",e.toString());
            }
        }
    }
}
```

다음은 이번 예제에서 사용되는 레이아웃 XML 파일인 “main.xml”이다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ImageView android:id="@+id/ReturnedImageView" android:layout_width="wrap_content" ↵
android:layout_height="wrap_content"></ImageView>
    <TextView android:layout_width="wrap_content" android:layout_height="wrap_content" ↵
android:text="Title:" android:id="@+id/TitleTextView"></TextView>
    <EditText android:layout_height="wrap_content" android:id="@+id/TitleEditText" ↵
android:layout_width="fill_parent"></EditText>
    <TextView android:layout_width="wrap_content" android:layout_height="wrap_content" ↵
android:text="Description" android:id="@+id/DescriptionTextView"></TextView>
    <EditText android:layout_height="wrap_content" android:layout_width="fill_parent" ↵
android:id="@+id/DescriptionEditText"></EditText>
    <Button android:layout_width="wrap_content" android:layout_height="wrap_content" ↵
android:id="@+id/TakePictureButton" android:text="Take Picture"></Button>
    <Button android:layout_width="wrap_content" android:layout_height="wrap_content" ↵
android:id="@+id/SaveDataButton" android:text="Save Data"></Button>
</LinearLayout>
```

이전 예제들에서와 마찬가지로 onActivityResult 메소드는 Camera 애플리케이션이 리턴하면 트리거된다. 새로 만들어진 이미지는 Bitmap으로 디코딩되어 표시된다. 이 버전에서는 적절한 사용자 인터페이스 요소들 또한 관리한다.

MediaStore를 사용하여 이미지 검색하기

안드로이드의 콘텐트 프로바이더가 발휘하는 힘은 갤러리 애플리케이션과 같은 것을 만들 때 진가가 드러난다. 콘텐트 프로바이더(이 경우에는 MediaStore)는 애플리케이션이 서로 공유하기 때문에, 이미지를 보여주는 애플리케이션을 만든다고 해서 카메라 애플리케이션을 따로 만든다거나 이미지를 저장하는 수단을 만든다든가 그렇게 할 이유가 없다. 애플리케이션들은 기본적으로 MediaStore를 사용하므로, 이를 활용하면 우리만의 갤러리 애플리케이션을 만들 수 있다.

MediaStore에서 선택하는 과정은 매우 직관적이다. 새로운 레코드를 만들 때 사용했던 URI를 그대로 사용하여 레코드들을 선택하면 된다.

```
Media.EXTERNAL_CONTENT_URI
```

사실 MediaStore를 비롯한 모든 콘텐트 프로바이더에서는 데이터베이스 작업이 별 차이 없다. 레코드를 선택하면 Cursor 객체를 받는데, 반복 연산에 이 객체를 사용한다.

우선 선택하는 과정에서는 리턴받기를 원하는 열들로 이뤄진 문자열 배열이 필요하다. MediaStore의 이미지에 쓸 표준 열은 MediaStore.Images.Media 클래스로 표현된다.

```
String[] columns = { Media.DATA, Media._ID, Media.TITLE, Media.DISPLAY_NAME };
```

실제 쿼리를 수행하기 위해 managedQuery 메소드를 사용한다. 첫 번째 인수는 URI고, 뒤이어 열 이름들로 구성된 배열이, 그 다음으로는 WHERE 절과 WHERE 절에 필요한 인수들, 마지막으로 ORDER BY 절이 이어진다.

다음은 지난 한 시간 동안 생성된 레코드들을 선택하여 가장 오래된 것부터 최신 것까지 늘어세운다.

먼저 oneHourAgo라는 변수부터 만들 텐데, 이 변수는 1970년 1월 1일부터 대략 한 시간 전까지 흘러간 시간을 초 단위로 담는다. 반면 System.currenTimeMillis()는 지나간 시간을 밀리초 단위로 리턴하는데, 기준 날짜는 마찬가지로 1970년 1월 1일이다. 따라서 이 숫자를 1,000으로 나누면 초 단위가 된다. 여기서 60분 × 60초를 빼면 한 시간 전

까지의 값이다.

```
long oneHourAgo = System.currentTimeMillis()/1000 - (60 * 60)
```

이제 이 값을 문자열 배열에 집어넣는다. 문자열 배열은 WHERE 절에 쓸 인수다.

```
String[] whereValues = {""+oneHourAgo};
```

그러고 나서 리턴받을 열을 선택한다.

```
String[] columns = { Media.DATA, Media._ID, Media.TITLE, Media.DISPLAY_NAME, Media.DATE_ADDED };
```

이제 쿼리를 수행한다. WHERE 절에는 물음표(?)가 하나 있는데, 이 문자는 다음 파라미터로 대체된다. 물음표가 여러 개면 넘겨받는 배열에도 해당 값이 여러 개다. 여기에 사용된 ORDER BY 절은 리턴된 데이터가 오름차순 날짜별로 정렬된다는 의미다.

```
cursor = managedQuery(Media.EXTERNAL_CONTENT_URI, columns, Media.DATE_ADDED + " > ?",↵
whereValues, Media.DATE_ADDED + " ASC");
```

물론 레코드를 모두 리턴받으려고 한다면 마지막 세 인수에 null을 넘겨줄 수도 있다.

```
Cursor cursor = managedQuery(Media.EXTERNAL_CONTENT_URI, columns, null, null, null);
```

리턴된 커서는 선택된 열 각각의 인덱스를 알려준다.

```
displayColumnIndex = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
```

커서에서 해당 필드를 선택하려면 이 인덱스가 필요하다. 우선 커서가 유효한 것인지, 결과를 가지고 있는지 확인해야 한다. 확인하려면 moveToFirst 메소드를 호출한다. 이 메소드는 커서가 결과를 하나도 담고 있지 않으면 거짓이 된다. 여기서는 Cursor 클래

스의 메소드 중 하나를 사용하여 실제 데이터를 선택한다. 사용되는 메소드는 해당 데이터의 타입에 따라 다른데, 문자열일 때는 getString, 정수일 때는 getInt 식이다.

```
if (cursor.moveToFirst()) {
    String displayName = cursor.getString(displayColumnIndex);
}
```

이미지 뷰어 애플리케이션 만들기

이어지는 내용은 예제에 대한 전체 코드로서, 이번 예제에서는 MediaStore에서 이미지를 찾는 쿼리를 실행하고, 슬라이드쇼 형태로 이미지를 하나씩 사용자에게 표시해준다.

```
package com.apress.proandroidmedia.ch1.mediastoregallery;

import android.app.Activity;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.provider.MediaStore;
import android.provider.MediaStore.Images.Media;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.TextView;

public class MediaStoreGallery extends Activity {

    public final static int DISPLAYWIDTH = 200;
    public final static int DISPLAYHEIGHT = 200;
```

화면의 크기로 이미지를 로드하여 표시하지 않고, 표시할 크기를 따로 정하려고 다음 상수들을 사용한다.

```
TextView titleTextView;
ImageButton imageButton;
```

이번 예제에서는 ImageView 대신 ImageButton을 사용한다. 그렇게 함으로써 Button(클릭될 수 있는 것)과 ImageView(이미지를 표시할 수 있는 것) 둘 다를 사용할 수 있다.

```
Cursor cursor;
Bitmap bmp;
String imageFilePath;
int fileColumn;
int titleColumn;
int displayColumn;

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

    titleTextView = (TextView) this.findViewById(R.id.TitleTextView);
    imageButton = (ImageButton) this.findViewById(R.id.ImageButton);
```

다음은 리턴받기를 원하는 열들을 지정하는 곳이다. 이때 반드시 문자열 배열 형태로 구성해야 한다. 이 배열을 managedQuery 메소드로 넘겨준다.

```
    String[] columns = { Media.DATA, Media._ID, Media.TITLE, Media.DISPLAY_NAME };
    cursor = managedQuery(Media.EXTERNAL_CONTENT_URI, columns, null, null, null);
```

Cursor 객체에서 데이터를 꺼내려면 각 열들의 인덱스를 알아야 한다. 여기서는 Media.DATA 대신 MediaStore.Images.Media.DATA를 사용할 텐데, 이를 통해 이 두 가지가 같은 것임을 보여주고자 한다. android.provider.MediaStore.Images.Media를 import 구문으로 미리 선언했기 때문에 Media.DATA는 줄여 쓰는 표현일 뿐이다.

```
    fileColumn = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
    titleColumn = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.TITLE);
    displayColumn = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DISPLAY_NAME);
```

쿼리를 실행하고 나서 결과 Cursor 객체를 가지면, 이 객체에 대해 moveToFirst를 호출하여 결과가 제대로 있는지 확인한다.

```
    if (cursor.moveToFirst()) {
        // titleTextView.setText(cursor.getString(titleColumn));
        titleTextView.setText(cursor.getString(displayColumn));

        imageFilePath = cursor.getString(fileColumn);
        bmp = getBitmap(imageFilePath);

        // 이미지를 표시한다.
        imageButton.setImageBitmap(bmp);
    }
```

이제 ImageButton에 쓸 OnClickListener를 새로 지정한다. OnClickListener는 Cursor 객체의 MoveToNext 메소드를 호출한다. 이 메소드는 결과들을 반복 처리하며 리턴된 이미지들을 끄집어내 표시한다.

```
    imageButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            if (cursor.moveToNext()) {
                // titleTextView.setText(cursor.getString(titleColumn));
                titleTextView.setText(cursor.getString(displayColumn));

                imageFilePath = cursor.getString(fileColumn);
                bmp = getBitmap(imageFilePath);
                imageButton.setImageBitmap(bmp);
            }
        }
    });
}
```

다음의 getBitmap이라는 메소드는 이미지의 크기 변환과 로드를 캡슐화하는 과정을 담당하는데, 앞에서 언급한 메모리 문제에 부딪히지 않고 이미지들을 표시하기 위해서는 이 과정이 필요하다.

```
    private Bitmap getBitmap(String imageFilePath) {
        // 이미지가 아니라 이미지의 치수를 로드한다.
        BitmapFactory.Options bmpFactoryOptions = new BitmapFactory.Options();
        bmpFactoryOptions.inJustDecodeBounds = true;
        Bitmap bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

        int heightRatio = (int) Math.ceil(bmpFactoryOptions.outHeight ↵
                / (float) DISPLAYHEIGHT);
        int widthRatio = (int) Math.ceil(bmpFactoryOptions.outWidth ↵
                / (float) DISPLAYWIDTH);

        Log.v("HEIGHTRATIO", "" + heightRatio);
        Log.v("WIDTHRATIO", "" + widthRatio);

        // 두 비율 다 1보다 크면 이미지의 가로, 세로 중 한쪽이 화면보다 크다.
        if (heightRatio > 1 && widthRatio > 1) {
            if (heightRatio > widthRatio) {
                // 높이 비율이 더 커서 그에 따라 맞춘다.
                bmpFactoryOptions.inSampleSize = heightRatio;
            } else {
                // 너비 비율이 더 커서 그에 따라 맞춘다.
                bmpFactoryOptions.inSampleSize = widthRatio;
            }
        }

        // 실제로 디코딩한다.
        bmpFactoryOptions.inJustDecodeBounds = false;
        bmp = BitmapFactory.decodeFile(imageFilePath, bmpFactoryOptions);

        return bmp;
    }
}
```

다음은 레이아웃 XML이다. 물론 res/layout/main.xml 파일에 들어가야 하는 코드다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    >
<ImageButton android:layout_width="wrap_content" android:layout_height="wrap_content" ↵
android:id="@+id/ImageButton"></ImageButton>
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/TitleTextView"
    android:text="Image Title"/>
</LinearLayout>
```

내부 메타데이터

교환 가능한 이미지 파일 포맷을 줄여 표현하는 EXIF는 이미지 파일 안에 메타데이터를 저장하는 표준 방법이다. EXIF 데이터를 지원하는 디지털 카메라용 애플리케이션이나 데스크톱 소프트웨어 애플리케이션은 수없이 많다. EXIF 데이터가 실제로 파일의 일부기 때문에, 이미지 파일을 한 곳에서 다른 곳으로 전송할 때 이 부분이 손실되면 안 된다. 예를 들어, 안드로이드 기기의 SD 카드에서 데스크톱 컴퓨터로 파일을 복사할 때 이 EXIF 데이터가 사라지는 것이 아니다. iPhoto와 같은 애플리케이션에서 이미지 파일을 열면, 해당 EXIF 데이터가 고스란히 보인다.

EXIF 데이터는 기술적 내용이 대부분이다. 태그들은 이미지를 캡처하는 것과 관련된 데이터를 주로 담는데, ExposureTime이나 ShutterSpeedValue를 그 예로 들 수 있다.

물론 이런 태그들 말고도 작성할 만한 의미 있는 태그들 또한 꽤 된다. 몇 가지 예를 들면 다음과 같다.

- UserComment: 사용자가 작성한 주석
- ImageDescription: 제목
- Artist: 이미지 저작자 또는 소유자
- Copyright: 이미지 저작권
- Software: 이미지를 생성한 소프트웨어

안드로이드는 EXIF 데이터를 읽고 쓸 수 있는 멋진 수단을 제공한다. 이때 사용되는 주 클래스가 ExifInterface다.

다음은 ExifInterface를 사용하여 특정 EXIF 데이터를 이미지 파일에서 읽는 방법이다.

```
ExifInterface ei = new ExifInterface(imageFilePath);
String imageDescription = ei.getAttribute("ImageDescription");
if (imageDescription != null)
{
    Log.v("EXIF", imageDescription);
}
```

그리고 다음은 ExifInterface를 사용하여 EXIF 데이터를 이미지 파일에 저장하는 방법이다.

```
ExifInterface ei = new ExifInterface(imageFilePath);
ei.setAttribute("ImageDescription","Something New");
```

ExifInterface에는 Camera 애플리케이션이 캡처한 이미지에 자동으로 들어가는 데이터들을 정의하는 상수들이 들었다.

EXIF 사양의 최신 버전은 2010년 4월에 나온 2.3이고, www.cipa.jp/english/hyoujunka/kikaku/pdf/DC-008-2010_E.pdf에서 확인할 수 있다.

정리

이 장에서 우리는 안드로이드가 제공하는 기본적인 이미지 캡처와 저장 방법을 살펴봤다. 안드로이드에서 제공하는 내장 Camera 애플리케이션의 강력한 힘을 체험했고, 인텐트를 통해 그 기능을 효과적으로 활용하는 방법 또한 경험했다. 그 과정에서 이미지 캡처 기능을 안드로이드 애플리케이션에 집어넣는 데 필요한 멋지고 일관된 인터페이스가 Camera 애플리케이션에 제공된다는 것을 알게 됐다.

큰 이미지를 다룰 때는 메모리 사용에 신경 써야 한다는 것도 살펴봤다. 메모리의 효율을 높이기 위해 크기가 조절된 이미지를 로드할 때 BitmapFactory 클래스가 필요하다. 스마트폰은 데스크톱 컴퓨터가 아니라는 사실에 신경 써야 한다.

우리는 안드로이드의 이미지용 내장 콘텐트 프로바이더인 MediaStore를 사용해봤다. 기기의 표준 위치에 이미지를 저장하는 방법을 배웠고, 캡처한 이미지를 활용하는 애플리케이션을 빠르게 빌드하기 위해 쿼리를 수행하는 방법도 배웠다.

마지막으로 EXIF라는 표준으로 메타데이터를 이미지에 연결하는 방법을 살펴봤다. EXIF는 다른 기기로 전송할 수도 있어 다양한 기기와 소프트웨어에 사용되고 있다. EXIF는 안드로이드의 미디어 기능을 파헤치는 그 시작점이 될 것이다.

이제부터 본격적으로 안드로이드의 미디어가 펼쳐진다!