Android Content Providers with Examples

In android, **Content Provider** will act as a central repository to store the data of the application in one place and make that data available for different applications to access whenever it's required.

In android, we can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.

Generally, the **Content Provider** is a part of an android application and it will act like more like a relational database to store the app data. We can perform multiple operations like insert, update, delete and edit on the data stored in content provider using **insert()**, **update()**, **delete()** and **query()** methods.

In android, we can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app.

In android, the content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.

We have different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

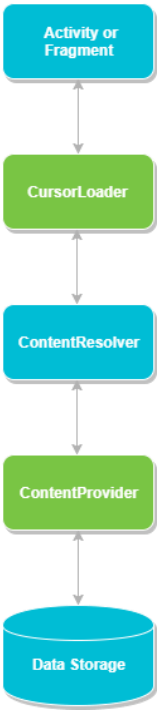
Access Data from Content Provider

To access a data from content provider, we need to use `ContentResolver` object in our application to communicate with the provider as a client. The `ContentResolver` object will communicate with the provider object (`ContentProvider`) which is implemented by an instance of class.

Generally, in android to send a request from UI to `ContentResolver` we have another object called **CursorLoader** which is used to run the query asynchronously in background. In android application, the UI components such as Activity or Fragment will call a **CursorLoader** to query and get a required data from `ContentProvider` using `ContentResolver`.

The `ContentProvider` object will receive data requests from the client, performs the requested actions (create, update, delete, retrieve) and return the result.

Following is the pictorial representation of requesting an operation from UI using Activity or Fragment to get the data from `ContentProvider` object.



This is how the interaction will happen between android application UI and content providers to perform the required actions to get data.

Content URIs

In android, **Content URI** is a URI that is used to query a content provider to get the required data. The Content URIs will contain the name of an entire-provider (**authority**) and the name that points to a table (**path**).

Generally the format of URI in android applications will be like as shown below

content://authority/path

Following are the details about various parts of an URI in android application.

content:// - The string **content://** is always present in the URI and it is used to represent the given URI is a content URI.

authority - It represents the name of content provider, for example phone, contacts, etc. and we need to use a fully qualified name for third party content providers like com.tutlane.contactprovider

path - It represents the table's path.

The `ContentResolver` object use the URI's **authority** to find the appropriate provider and send the query objects to the correct provider. After that `ContentProvider` uses the **path** of content URI to choose the right table to access.

Following is the example of a simple example of URI in android applications.

content://contacts_info/users

Here the string **content://** is used to represent URI is a content URI, **contacts_info** string is the name of the provider's authority and **users** string is the table's path.

Creating a Content Provider

To create a content provider in android applications we should follow below steps.

- We need to create a content provider class that extends the ContentProvider base class.
- We need to define our content provider URI to access the content.
- The `ContentProvider` class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.
- We need to register our content provider in AndroidManifest.xml using **<provider>** tag.

Following are the list of methods which need to implement as a part of `ContentProvider` class.

- **query()** - It receives a request from the client. By using arguments it will get a data from the requested table and return the data as a **Cursor** object.
- **insert()** - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.
- **update()** - This method will update existing rows in our content provider and it returns the number of rows updated.
- **delete()** - This method will delete the rows in our content provider and it returns the number of rows deleted.
- **getType()** - This method will return the MIME type of data to given content URI.
- **onCreate()** - This method will initialize our provider. The android system will call this method immediately after it creates our provider.

Android Content Provider Example

Following is the example of using **Content Provider** in android applications. Here we will create our own content provider to insert and access data in android application.

Create a new android application using android studio and give names as `ContentProvider`.

Now we need to create our own content provider file **UserProvider.java** in **\src\main\java\com.tutlane.contentprovider** path to define our actual provider and associated methods for that right-click on your application folder → Go to **New** → select **Java Class** and give name as **UserProvider.java**.

Once we create a new file **UserProvider.java**, open it and write the code like as shown below

UserProvider.java

```
package com.tutlane.contentprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import java.util.HashMap;

/**
 * Created by sureshdasari on 29-07-2017.
 */

public class UsersProvider extends ContentProvider {
   static final String PROVIDER_NAME = "com.tutlane.contentprovider.UserProvider";
   static final String URL = "content://" + PROVIDER_NAME + "/users";
   static final Uri CONTENT_URI = Uri.parse(URL);

   static final String id = "id";
   static final String name = "name";
   static final int uriCode = 1;
   static final UriMatcher uriMatcher;
   private static HashMap<String, String> values;
   static {
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(PROVIDER_NAME, "users", uriCode);
      uriMatcher.addURI(PROVIDER_NAME, "users/*", uriCode);
   }

  @Override
   public String getType(Uri uri) {
```

```
    switch (uriMatcher.match(uri)) {
        case uriCode:
            return "vnd.android.cursor.dir/users";
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}

@Override
public boolean onCreate() {
    Context context = getContext();
    DatabaseHelper dbHelper = new DatabaseHelper(context);
    db = dbHelper.getWritableDatabase();
    if (db != null) {
        return true;
    }
    return false;
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(TABLE_NAME);

    switch (uriMatcher.match(uri)) {
        case uriCode:
            qb.setProjectionMap(values);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    if (sortOrder == null || sortOrder == "") {
        sortOrder = id;
    }
    Cursor c = qb.query(db, projection, selection, selectionArgs, null,
            null, sortOrder);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
@Override
public Uri insert(Uri uri, ContentValues values) {
    long rowID = db.insert(TABLE_NAME, "", values);
    if (rowID > 0) {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLiteException("Failed to add a record into " + uri);
}
@Override
```

```
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)) {
            case uriCode:
                count = db.update(TABLE_NAME, values, selection, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)) {
            case uriCode:
                count = db.delete(TABLE_NAME, selection, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
    private SQLiteDatabase db;
    static final String DATABASE_NAME = "EmpDB";
    static final String TABLE_NAME = "Employees";
    static final int DATABASE_VERSION = 1;
    static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME
            + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + " name TEXT NOT NULL);";

    private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE_DB_TABLE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
            onCreate(db);
        }
    }
}
```

Now open an **activity_main.xml** file from **\src\main\res\layout** path and write the code like as shown below.

activity_main.xml

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

<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Name"
      android:layout_marginLeft="100dp"
      android:layout_marginTop="100dp"/>
   <EditText
      android:id="@+id/txtName"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:ems="10"/>
   <Button
      android:id="@+id/btnAdd"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="onClickAddDetails"
      android:layout_marginLeft="100dp"
      android:text="Add User"/>

   <Button
      android:id="@+id/btnRetrieve"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="onClickShowDetails"
      android:layout_marginLeft="100dp"
      android:text="Show Users"/>
   <TextView
      android:id="@+id/res"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:clickable="false"
      android:ems="10"/>
</LinearLayout>
```

Now open **MainActivity.java** file and write the code like as shown below

MainActivity.java

```
package com.tutlane.contentprovider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        InputMethodManager imm =
(InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(), 0);
        return true;
    }
    public void onClickAddDetails(View view) {
        ContentValues values = new ContentValues();
        values.put(UsersProvider.name, ((EditText) findViewById(R.id.txtName)).getText().toString());
        getContentResolver().insert(UsersProvider.CONTENT_URI, values);
        Toast.makeText(getBaseContext(), "New Record Inserted", Toast.LENGTH_LONG).show();
    }

    public void onClickShowDetails(View view) {
        // Retrieve employee records
        TextView resultView= (TextView) findViewById(R.id.res);
        Cursor cursor =
getContentResolver().query(Uri.parse("content://com.tutlane.contentprovider.UserProvider/users"), null, null,
null, null);
        if(cursor.moveToFirst()) {
            StringBuilder strBuild=new StringBuilder();
            while (!cursor.isAfterLast()) {
                strBuild.append("\n"+cursor.getString(cursor.getColumnIndex("id"))+ "-"+
cursor.getString(cursor.getColumnIndex("name")));
                cursor.moveToNext();
            }
            resultView.setText(strBuild);
        }
```

```
        else {
            resultView.setText("No Records Found");
        }
    }
}
```

We need to include this newly created content provider in android manifest file (**ActivityManifest.xml**) using **<provider>** attribute like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.tutlane.contentprovider">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider
            android:authorities="com.tutlane.contentprovider.UserProvider"
            android:name=".UsersProvider">
            </provider>
    </application>
</manifest>
```

Output of Android Content Provider Example

When we run above example in android emulator we will get a result like as shown below

This is how we can use content providers in our android application to store the data in a centralizing location to allow other apps to access based on our requirements.