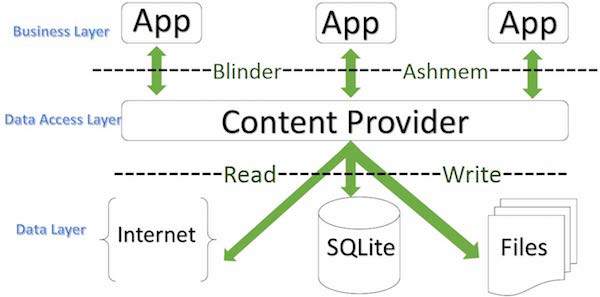
**Android - Content Providers**

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



**ContentProvider**

**sometimes it is required to share data across applications. This is where content providers become very useful.**

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

public class My Application extends ContentProvider {

}

**Content URIs**

To query a content provider, you specify the query string in the form of a URI which has following format −

<prefix>://<authority>/<data\_type>/<id>

Here is the detail of various parts of the URI −

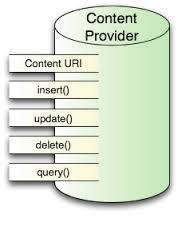
|  |  |
| --- | --- |
| **Sr.No** | **Part & Description** |
| 1 | **prefix**  This is always set to content:// |
| 2 | **authority**  This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider* |
| 3 | **data\_type**  This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people* |
| 4 | **id**  This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*. |

**Create Content Provider**

This involves number of simple steps to create your own content provider.

* First of all you need to create a Content Provider class that extends the *ContentProviderbaseclass.*
* Second, you need to define your content provider URI address which will be used to access the content.
* Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
* Next you will have to implement Content Provider queries to perform different database specific operations.
* Finally register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working −



**ContentProvider**

* **onCreate()** This method is called when the provider is started.
* **query()** This method receives a request from a client. The result is returned as a Cursor object.
* **insert()**This method inserts a new record into the content provider.
* **delete()** This method deletes an existing record from the content provider.
* **update()** This method updates an existing record from the content provider.
* **getType()** This method returns the MIME type of the data at the given URI.

**Example**

This example will explain you how to create your own *ContentProvider*. So let's follow the following steps to similar to what we followed while creating *Hello World Example*−

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.MyApplication*, with blank Activity. |
| 2 | Modify main activity file *MainActivity.java* to add two new methods *onClickAddName()* and *onClickRetrieveStudents()*. |
| 3 | Create a new java file called *StudentsProvider.java* under the package *com.example.MyApplication* to define your actual provider and associated methods. |
| 4 | Register your content provider in your *AndroidManifest.xml* file using <provider.../> tag |
| 5 | Modify the default content of *res/layout/activity\_main.xml* file to include a small GUI to add students records. |
| 6 | No need to change string.xml.Android studio take care of string.xml file. |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **src/com.example.MyApplication/MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added two new methods *onClickAddName()* and *onClickRetrieveStudents()* to handle user interaction with the application.

package com.example.MyApplication;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.ContentValues;

import android.content.CursorLoader;

import android.database.Cursor;

import android.view.Menu;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void onClickAddName(View view) {

// Add a new student record

ContentValues values = new ContentValues();

values.put(StudentsProvider.NAME,

((EditText)findViewById(R.id.editText2)).getText().toString());

values.put(StudentsProvider.GRADE,

((EditText)findViewById(R.id.editText3)).getText().toString());

Uri uri = getContentResolver().insert(

StudentsProvider.CONTENT\_URI, values);

Toast.makeText(getBaseContext(),

uri.toString(), Toast.LENGTH\_LONG).show();

}

public void onClickRetrieveStudents(View view) {

// Retrieve student records

String URL = "content://com.example.MyApplication.StudentsProvider";

Uri students = Uri.parse(URL);

Cursor c = managedQuery(students, null, null, null, "name");

if (c.moveToFirst()) {

do{

Toast.makeText(this,

c.getString(c.getColumnIndex(StudentsProvider.\_ID)) +

", " + c.getString(c.getColumnIndex( StudentsProvider.NAME)) +

", " + c.getString(c.getColumnIndex( StudentsProvider.GRADE)),

Toast.LENGTH\_SHORT).show();

} while (c.moveToNext());

}

}

}

Create new file StudentsProvider.java under *com.example.MyApplication* package and following is the content of **src/com.example.MyApplication/StudentsProvider.java** −

package com.example.MyApplication;

import java.util.HashMap;

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.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteQueryBuilder;

import android.net.Uri;

import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {

static final String PROVIDER\_NAME = "com.example.MyApplication.StudentsProvider";

static final String URL = "content://" + PROVIDER\_NAME + "/students";

static final Uri CONTENT\_URI = Uri.parse(URL);

static final String \_ID = "\_id";

static final String NAME = "name";

static final String GRADE = "grade";

private static HashMap<String, String> STUDENTS\_PROJECTION\_MAP;

static final int STUDENTS = 1;

static final int STUDENT\_ID = 2;

static final UriMatcher uriMatcher;

static{

uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

uriMatcher.addURI(PROVIDER\_NAME, "students", STUDENTS);

uriMatcher.addURI(PROVIDER\_NAME, "students/#", STUDENT\_ID);

}

/\*\*

\* Database specific constant declarations

\*/

private SQLiteDatabase db;

static final String DATABASE\_NAME = "College";

static final String STUDENTS\_TABLE\_NAME = "students";

static final int DATABASE\_VERSION = 1;

static final String CREATE\_DB\_TABLE =

" CREATE TABLE " + STUDENTS\_TABLE\_NAME +

" (\_id INTEGER PRIMARY KEY AUTOINCREMENT, " +

" name TEXT NOT NULL, " +

" grade TEXT NOT NULL);";

/\*\*

\* Helper class that actually creates and manages

\* the provider's underlying data repository.

\*/

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 " + STUDENTS\_TABLE\_NAME);

onCreate(db);

}

}

@Override

public boolean onCreate() {

Context context = getContext();

DatabaseHelper dbHelper = new DatabaseHelper(context);

/\*\*

\* Create a write able database which will trigger its

\* creation if it doesn't already exist.

\*/

db = dbHelper.getWritableDatabase();

return (db == null)? false:true;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

/\*\*

\* Add a new student record

\*/

long rowID = db.insert( STUDENTS\_TABLE\_NAME, "", values);

/\*\*

\* If record is added successfully

\*/

if (rowID > 0) {

Uri \_uri = ContentUris.withAppendedId(CONTENT\_URI, rowID);

getContext().getContentResolver().notifyChange(\_uri, null);

return \_uri;

}

throw new SQLException("Failed to add a record into " + uri);

}

@Override

public Cursor query(Uri uri, String[] projection,

String selection,String[] selectionArgs, String sortOrder) {

SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

qb.setTables(STUDENTS\_TABLE\_NAME);

switch (uriMatcher.match(uri)) {

case STUDENTS:

qb.setProjectionMap(STUDENTS\_PROJECTION\_MAP);

break;

case STUDENT\_ID:

qb.appendWhere( \_ID + "=" + uri.getPathSegments().get(1));

break;

default:

}

if (sortOrder == null || sortOrder == ""){

/\*\*

\* By default sort on student names

\*/

sortOrder = NAME;

}

Cursor c = qb.query(db, projection, selection,

selectionArgs,null, null, sortOrder);

/\*\*

\* register to watch a content URI for changes

\*/

c.setNotificationUri(getContext().getContentResolver(), uri);

return c;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

int count = 0;

switch (uriMatcher.match(uri)){

case STUDENTS:

count = db.delete(STUDENTS\_TABLE\_NAME, selection, selectionArgs);

break;

case STUDENT\_ID:

String id = uri.getPathSegments().get(1);

count = db.delete( STUDENTS\_TABLE\_NAME, \_ID + " = " + id +

(!TextUtils.isEmpty(selection) ? "

AND (" + selection + ')' : ""), selectionArgs);

break;

default:

throw new IllegalArgumentException("Unknown URI " + uri);

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

@Override

public int update(Uri uri, ContentValues values,

String selection, String[] selectionArgs) {

int count = 0;

switch (uriMatcher.match(uri)) {

case STUDENTS:

count = db.update(STUDENTS\_TABLE\_NAME, values, selection, selectionArgs);

break;

case STUDENT\_ID:

count = db.update(STUDENTS\_TABLE\_NAME, values,

\_ID + " = " + uri.getPathSegments().get(1) +

(!TextUtils.isEmpty(selection) ? "

AND (" +selection + ')' : ""), selectionArgs);

break;

default:

throw new IllegalArgumentException("Unknown URI " + uri );

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

@Override

public String getType(Uri uri) {

switch (uriMatcher.match(uri)){

/\*\*

\* Get all student records

\*/

case STUDENTS:

return "vnd.android.cursor.dir/vnd.example.students";

/\*\*

\* Get a particular student

\*/

case STUDENT\_ID:

return "vnd.android.cursor.item/vnd.example.students";

default:

throw new IllegalArgumentException("Unsupported URI: " + uri);

}

}

}

Following will the modified content of *AndroidManifest.xml* file. Here we have added <provider.../> tag to include our content provider:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.MyApplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

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:name="StudentsProvider"

android:authorities="com.example.MyApplication.StudentsProvider"/>

</application>

</manifest>

Following will be the content of **res/layout/activity\_main.xml** file−

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context="com.example.MyApplication.MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Content provider"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point "

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_below="@+id/textView1"

android:layout\_centerHorizontal="true" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_below="@+id/textView2"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/button2"

android:text="Add Name"

android:layout\_below="@+id/editText3"

android:layout\_alignRight="@+id/textView2"

android:layout\_alignEnd="@+id/textView2"

android:layout\_alignLeft="@+id/textView2"

android:layout\_alignStart="@+id/textView2"

android:onClick="onClickAddName"/>

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText"

android:layout\_below="@+id/imageButton"

android:layout\_alignRight="@+id/imageButton"

android:layout\_alignEnd="@+id/imageButton" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText2"

android:layout\_alignTop="@+id/editText"

android:layout\_alignLeft="@+id/textView1"

android:layout\_alignStart="@+id/textView1"

android:layout\_alignRight="@+id/textView1"

android:layout\_alignEnd="@+id/textView1"

android:hint="Name"

android:textColorHint="@android:color/holo\_blue\_light" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText3"

android:layout\_below="@+id/editText"

android:layout\_alignLeft="@+id/editText2"

android:layout\_alignStart="@+id/editText2"

android:layout\_alignRight="@+id/editText2"

android:layout\_alignEnd="@+id/editText2"

android:hint="Grade"

android:textColorHint="@android:color/holo\_blue\_bright" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Retrive student"

android:id="@+id/button"

android:layout\_below="@+id/button2"

android:layout\_alignRight="@+id/editText3"

android:layout\_alignEnd="@+id/editText3"

android:layout\_alignLeft="@+id/button2"

android:layout\_alignStart="@+id/button2"

android:onClick="onClickRetrieveStudents"/>

</RelativeLayout>

Make sure you have following content of **res/values/strings.xml** file:

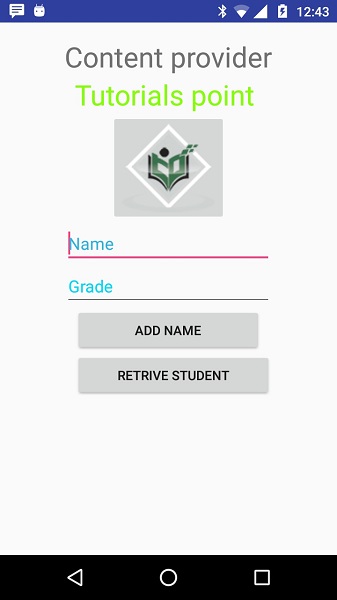
<?xml version="1.0" encoding="utf-8"?>

<resources>

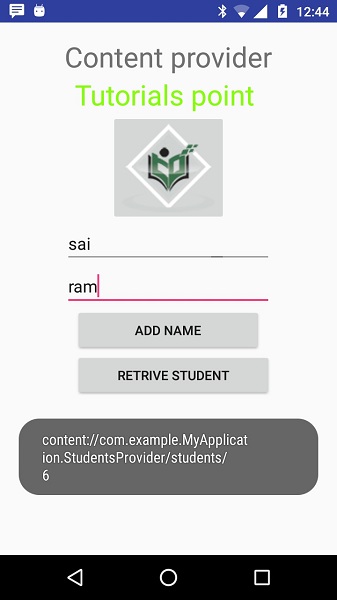
<string name="app\_name">My Application</string>

</resources>;

Let's try to run our modified **My Application** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android Studio IDE, open one of your project's activity files and click Run Android StudioRun Iconicon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window, be patience because it may take sometime based on your computer speed −



Now let's enter student **Name** and **Grade** and finally click on **Add Name** button, this will add student record in the database and will flash a message at the bottom showing ContentProvider URI along with record number added in the database. This operation makes use of our **insert()** method. Let's repeat this process to add few more students in the database of our content provider.



Once you are done with adding records in the database, now its time to ask ContentProvider to give us those records back, so let's click **Retrieve Students** button which will fetch and display all the records one by one which is as per our the implementation of our **query()** method.

You can write activities against update and delete operations by providing callback functions in **MainActivity.java** file and then modify user interface to have buttons for update and deleted operations in the same way as we have done for add and read operations.

This way you can use existing Content Provider like Address Book or you can use Content Provider concept in developing nice database oriented applications where you can perform all sort of database operations like read, write, update and delete as explained above in the example.

# Creating a content provider

A content provider manages access to a central repository of data. You implement a provider as one or more classes in an Android application, along with elements in the manifest file. One of your classes implements a subclass [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html), which is the interface between your provider and other applications. Although content providers are meant to make data available to other applications, you may of course have activities in your application that allow the user to query and modify the data managed by your provider.

The rest of this topic is a basic list of steps for building a content provider and a list of APIs to use.

## Before you start building

Before you start building a provider, consider the following:

1. **Decide if you need a content provider**. You need to build a content provider if you want to provide one or more of the following features:
   * You want to offer complex data or files to other applications.
   * You want to allow users to copy complex data from your app into other apps.
   * You want to provide custom search suggestions using the search framework.
   * You want to expose your application data to widgets.
   * You want to implement the [AbstractThreadedSyncAdapter](https://developer.android.com/reference/android/content/AbstractThreadedSyncAdapter.html), [CursorAdapter](https://developer.android.com/reference/android/widget/CursorAdapter.html), or [CursorLoader](https://developer.android.com/reference/android/content/CursorLoader.html) classes.

You don't need a provider to use databases or other types of persistent storage if the use is entirely within your own application and you don’t need any of the features listed above. Instead, you can use one of the storage systems described on the [Saving App Data](https://developer.android.com/training/data-storage/index.html) page.

1. If you haven't done so already, read the topic [Content provider basics](https://developer.android.com/guide/topics/providers/content-provider-basics.html) to learn more about providers and how they work.

Next, follow these steps to build your provider:

1. Design the raw storage for your data. A content provider offers data in two ways:

File data

Data that normally goes into files, such as photos, audio, or videos. Store the files in your application's private space. In response to a request for a file from another application, your provider can offer a handle to the file.

"Structured" data

Data that normally goes into a database, array, or similar structure. Store the data in a form that's compatible with tables of rows and columns. A row represents an entity, such as a person or an item in inventory. A column represents some data for the entity, such a person's name or an item's price. A common way to store this type of data is in an SQLite database, but you can use any type of persistent storage. To learn more about the storage types available in the Android system, see the section [Designing data storage](https://developer.android.com/guide/topics/providers/content-provider-creating#DataStorage).

1. Define a concrete implementation of the [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) class and its required methods. This class is the interface between your data and the rest of the Android system. For more information about this class, see the section [Implementing the ContentProvider class](https://developer.android.com/guide/topics/providers/content-provider-creating#ContentProvider).
2. Define the provider's authority string, its content URIs, and column names. If you want the provider's application to handle intents, also define intent actions, extras data, and flags. Also define the permissions that you will require for applications that want to access your data. You should consider defining all of these values as constants in a separate contract class; later, you can expose this class to other developers. For more information about content URIs, see the section [Designing Content URIs](https://developer.android.com/guide/topics/providers/content-provider-creating#ContentURI). For more information about intents, see the section [Intents and Data Access](https://developer.android.com/guide/topics/providers/content-provider-creating#Intents).
3. Add other optional pieces, such as sample data or an implementation of [AbstractThreadedSyncAdapter](https://developer.android.com/reference/android/content/AbstractThreadedSyncAdapter.html) that can synchronize data between the provider and cloud-based data.

## Designing data storage

A content provider is the interface to data saved in a structured format. Before you create the interface, you must decide how to store the data. You can store the data in any form you like, and then design the interface to read and write the data as necessary.

These are some of the data storage technologies that are available in Android:

* If you are working with structured data then consider either a relational database such as SQLite, or a non relational key-value datastore such as [LevelDB](https://github.com/google/leveldb). If you are working with unstructured data such as audio, image, or video media then consider storing the data as files. You can mix and match several different types of storage, and expose them using a single content provider if necessary.
* The Android system can interact with the Room persistence library, which provides access to the SQLite database API that Android's own providers use to store table-oriented data. To create a database using this library, instantiate a subclass of [RoomDatabase](https://developer.android.com/reference/android/arch/persistence/room/RoomDatabase.html), as described in the [Room](https://developer.android.com/training/data-storage/room/index.html) persistence library guide.

Remember that you don't have to use a database to implement your repository. A provider appears externally as a set of tables, similar to a relational database, but this is not a requirement for the provider's internal implementation.

* For storing file data, Android has a variety of file-oriented APIs. To learn more about file storage, read the topic [Data Storage](https://developer.android.com/guide/topics/data/data-storage.html). If you're designing a provider that offers media-related data such as music or videos, you can have a provider that combines table data and files.
* In rare cases, you might benefit from implementing more than one content provider for a single application. For example, you might want to share some data with a widget using one content provider, and expose a different set of data for sharing with other applications.
* For working with network-based data, use classes in [java.net](https://developer.android.com/reference/java/net/package-summary.html) and [android.net](https://developer.android.com/reference/android/net/package-summary.html). You can also synchronize network-based data to a local data store such as a database, and then offer the data as tables or files. The [Basic Sync Adapter](https://developer.android.com/samples/BasicSyncAdapter/index.html) sample application demonstrates this type of synchronization.

**Note**: If you make a change to your repository that isn't backwards-compatible, you need to mark the repository with a new version number. You also need to increase the version number for your app that implements the new content provider. Making this change prevents system downgrades from causing the system to crash when it attempts to reinstall an app that has an incompatible content provider.

### Data design considerations

Here are some tips for designing your provider's data structure:

* Table data should always have a "primary key" column that the provider maintains as a unique numeric value for each row. You can use this value to link the row to related rows in other tables (using it as a "foreign key"). Although you can use any name for this column, using [BaseColumns.\_ID](https://developer.android.com/reference/android/provider/BaseColumns.html#_ID) is the best choice, because linking the results of a provider query to a [ListView](https://developer.android.com/reference/android/widget/ListView.html) requires one of the retrieved columns to have the name \_ID.
* If you want to provide bitmap images or other very large pieces of file-oriented data, store the data in a file and then provide it indirectly rather than storing it directly in a table. If you do this, you need to tell users of your provider that they need to use a [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) file method to access the data.
* Use the Binary Large OBject (BLOB) data type to store data that varies in size or has a varying structure. For example, you can use a BLOB column to store a [protocol buffer](http://code.google.com/p/protobuf) or [JSON structure](http://www.json.org).

You can also use a BLOB to implement a schema-independent table. In this type of table, you define a primary key column, a MIME type column, and one or more generic columns as BLOB. The meaning of the data in the BLOB columns is indicated by the value in the MIME type column. This allows you to store different row types in the same table. The Contacts Provider's "data" table [ContactsContract.Data](https://developer.android.com/reference/android/provider/ContactsContract.Data.html) is an example of a schema-independent table.

## Designing content URIs

A **content URI** is a URI that identifies data in a provider. Content URIs include the symbolic name of the entire provider (its **authority**) and a name that points to a table or file (a **path**). The optional id part points to an individual row in a table. Every data access method of [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) has a content URI as an argument; this allows you to determine the table, row, or file to access.

The basics of content URIs are described in the topic [Content provider basics](https://developer.android.com/guide/topics/providers/content-provider-basics.html).

### Designing an authority

A provider usually has a single authority, which serves as its Android-internal name. To avoid conflicts with other providers, you should use Internet domain ownership (in reverse) as the basis of your provider authority. Because this recommendation is also true for Android package names, you can define your provider authority as an extension of the name of the package containing the provider. For example, if your Android package name is com.example.<appname>, you should give your provider the authority com.example.<appname>.provider.

### Designing a path structure

Developers usually create content URIs from the authority by appending paths that point to individual tables. For example, if you have two tables table1 and table2, you combine the authority from the previous example to yield the content URIs com.example.<appname>.provider/table1 and com.example.<appname>.provider/table2. Paths aren't limited to a single segment, and there doesn't have to be a table for each level of the path.

### Handling content URI IDs

By convention, providers offer access to a single row in a table by accepting a content URI with an ID value for the row at the end of the URI. Also by convention, providers match the ID value to the table's \_ID column, and perform the requested access against the row that matches.

This convention facilitates a common design pattern for apps accessing a provider. The app does a query against the provider and displays the resulting [Cursor](https://developer.android.com/reference/android/database/Cursor.html) in a [ListView](https://developer.android.com/reference/android/widget/ListView.html) using a [CursorAdapter](https://developer.android.com/reference/android/widget/CursorAdapter.html). The definition of [CursorAdapter](https://developer.android.com/reference/android/widget/CursorAdapter.html) requires one of the columns in the [Cursor](https://developer.android.com/reference/android/database/Cursor.html) to be \_ID

The user then picks one of the displayed rows from the UI in order to look at or modify the data. The app gets the corresponding row from the [Cursor](https://developer.android.com/reference/android/database/Cursor.html) backing the [ListView](https://developer.android.com/reference/android/widget/ListView.html), gets the \_ID value for this row, appends it to the content URI, and sends the access request to the provider. The provider can then do the query or modification against the exact row the user picked.

### Content URI patterns

To help you choose which action to take for an incoming content URI, the provider API includes the convenience class [UriMatcher](https://developer.android.com/reference/android/content/UriMatcher.html), which maps content URI "patterns" to integer values. You can use the integer values in a switch statement that chooses the desired action for the content URI or URIs that match a particular pattern.

A content URI pattern matches content URIs using wildcard characters:

* **\*:** Matches a string of any valid characters of any length.
* **#:** Matches a string of numeric characters of any length.

As an example of designing and coding content URI handling, consider a provider with the authority com.example.app.provider that recognizes the following content URIs pointing to tables:

* content://com.example.app.provider/table1: A table called table1.
* content://com.example.app.provider/table2/dataset1: A table called dataset1.
* content://com.example.app.provider/table2/dataset2: A table called dataset2.
* content://com.example.app.provider/table3: A table called table3.

The provider also recognizes these content URIs if they have a row ID appended to them, as for example content://com.example.app.provider/table3/1 for the row identified by 1 in table3.

The following content URI patterns would be possible:

content://com.example.app.provider/\*

Matches any content URI in the provider.

content://com.example.app.provider/table2/\*:

Matches a content URI for the tables dataset1 and dataset2, but doesn't match content URIs for table1 or table3.

content://com.example.app.provider/table3/#: Matches a content URI for single rows in table3, such as content://com.example.app.provider/table3/6 for the row identified by 6.

The following code snippet shows how the methods in [UriMatcher](https://developer.android.com/reference/android/content/UriMatcher.html) work. This code handles URIs for an entire table differently from URIs for a single row, by using the content URI pattern content://<authority>/<path> for tables, and content://<authority>/<path>/<id> for single rows.

The method [addURI()](https://developer.android.com/reference/android/content/UriMatcher.html#addURI(java.lang.String, java.lang.String, int)) maps an authority and path to an integer value. The method [match()](https://developer.android.com/reference/android/content/UriMatcher.html#match(android.net.Uri)) returns the integer value for a URI. A switch statement chooses between querying the entire table, and querying for a single record:

[Kotlin](https://developer.android.com/guide/topics/providers/content-provider-creating#kotlin)[Java](https://developer.android.com/guide/topics/providers/content-provider-creating#java)

private val sUriMatcher = UriMatcher(UriMatcher.NO\_MATCH).apply {  
    /\*  
     \* The calls to addURI() go here, for all of the content URI patterns that the provider  
     \* should recognize. For this snippet, only the calls for table 3 are shown.  
     \*/  
  
    /\*  
     \* Sets the integer value for multiple rows in table 3 to 1. Notice that no wildcard is used  
     \* in the path  
     \*/  
    addURI("com.example.app.provider", "table3", 1)  
  
    /\*  
     \* Sets the code for a single row to 2. In this case, the "#" wildcard is  
     \* used. "content://com.example.app.provider/table3/3" matches, but  
     \* "content://com.example.app.provider/table3 doesn't.  
     \*/  
    addURI("com.example.app.provider", "table3/#", 2)  
}  
...  
class ExampleProvider : ContentProvider() {  
    ...  
    // Implements ContentProvider.query()  
    override fun query(  
            uri: Uri?,  
            projection: Array<out String>?,  
            selection: String?,  
            selectionArgs: Array<out String>?,  
            sortOrder: String?  
    ): Cursor? {  
        var localSortOrder: String = sortOrder ?: ""  
        var localSelection: String = selection ?: ""  
        when (sUriMatcher.match(uri)) {  
            1 -> { // If the incoming URI was for all of table3  
                if (localSortOrder.isEmpty()) {  
                    localSortOrder = "\_ID ASC"  
                }  
            }  
            2 -> {  // If the incoming URI was for a single row  
                /\*  
                 \* Because this URI was for a single row, the \_ID value part is  
                 \* present. Get the last path segment from the URI; this is the \_ID value.  
                 \* Then, append the value to the WHERE clause for the query  
                 \*/  
                localSelection += "\_ID ${uri?.lastPathSegment}"  
            }  
            else -> { // If the URI is not recognized  
                // You should do some error handling here.  
            }  
        }  
  
        // call the code to actually do the query  
    }  
}

Another class, [ContentUris](https://developer.android.com/reference/android/content/ContentUris.html), provides convenience methods for working with the id part of content URIs. The classes [Uri](https://developer.android.com/reference/android/net/Uri.html) and [Uri.Builder](https://developer.android.com/reference/android/net/Uri.Builder.html) include convenience methods for parsing existing [Uri](https://developer.android.com/reference/android/net/Uri.html) objects and building new ones.

## Implementing the ContentProvider class

The [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) instance manages access to a structured set of data by handling requests from other applications. All forms of access eventually call [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html), which then calls a concrete method of [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) to get access.

### Required methods

The abstract class [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) defines six abstract methods that you must implement as part of your own concrete subclass. All of these methods except [onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate()) are called by a client application that is attempting to access your content provider:

[query()](https://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String))

Retrieve data from your provider. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result. Return the data as a [Cursor](https://developer.android.com/reference/android/database/Cursor.html) object.

[insert()](https://developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues))

Insert a new row into your provider. Use the arguments to select the destination table and to get the column values to use. Return a content URI for the newly-inserted row.

[update()](https://developer.android.com/reference/android/content/ContentProvider.html#update(android.net.Uri, android.content.ContentValues, java.lang.String, java.lang.String[]))

Update existing rows in your provider. Use the arguments to select the table and rows to update and to get the updated column values. Return the number of rows updated.

[delete()](https://developer.android.com/reference/android/content/ContentProvider.html#delete(android.net.Uri, java.lang.String, java.lang.String[]))

Delete rows from your provider. Use the arguments to select the table and the rows to delete. Return the number of rows deleted.

[getType()](https://developer.android.com/reference/android/content/ContentProvider.html#getType(android.net.Uri))

Return the MIME type corresponding to a content URI. This method is described in more detail in the section [Implementing content provider MIME types](https://developer.android.com/guide/topics/providers/content-provider-creating#MIMETypes).

[onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate())

Initialize your provider. The Android system calls this method immediately after it creates your provider. Notice that your provider is not created until a [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) object tries to access it.

Notice that these methods have the same signature as the identically-named [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) methods.

Your implementation of these methods should account for the following:

* All of these methods except [onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate()) can be called by multiple threads at once, so they must be thread-safe. To learn more about multiple threads, see the topic [Processes and Threads](https://developer.android.com/guide/components/processes-and-threads.html).
* Avoid doing lengthy operations in [onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate()). Defer initialization tasks until they are actually needed. The section [Implementing the onCreate() method](https://developer.android.com/guide/topics/providers/content-provider-creating#OnCreate) discusses this in more detail.
* Although you must implement these methods, your code does not have to do anything except return the expected data type. For example, you may want to prevent other applications from inserting data into some tables. To do this, you can ignore the call to [insert()](https://developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues)) and return 0.

### Implementing the query() method

The [ContentProvider.query()](https://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)) method must return a [Cursor](https://developer.android.com/reference/android/database/Cursor.html) object, or if it fails, throw an [Exception](https://developer.android.com/reference/java/lang/Exception.html). If you are using an SQLite database as your data storage, you can simply return the [Cursor](https://developer.android.com/reference/android/database/Cursor.html) returned by one of the query() methods of the [SQLiteDatabase](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html) class. If the query does not match any rows, you should return a [Cursor](https://developer.android.com/reference/android/database/Cursor.html) instance whose [getCount()](https://developer.android.com/reference/android/database/Cursor.html#getCount()) method returns 0. You should return null only if an internal error occurred during the query process.

If you aren't using an SQLite database as your data storage, use one of the concrete subclasses of [Cursor](https://developer.android.com/reference/android/database/Cursor.html). For example, the [MatrixCursor](https://developer.android.com/reference/android/database/MatrixCursor.html) class implements a cursor in which each row is an array of [Object](https://developer.android.com/reference/java/lang/Object.html). With this class, use [addRow()](https://developer.android.com/reference/android/database/MatrixCursor.html#addRow(java.lang.Object[])) to add a new row.

Remember that the Android system must be able to communicate the [Exception](https://developer.android.com/reference/java/lang/Exception.html) across process boundaries. Android can do this for the following exceptions that may be useful in handling query errors:

* [IllegalArgumentException](https://developer.android.com/reference/java/lang/IllegalArgumentException.html) (You may choose to throw this if your provider receives an invalid content URI)
* [NullPointerException](https://developer.android.com/reference/java/lang/NullPointerException.html)

### Implementing the insert() method

The [insert()](https://developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues)) method adds a new row to the appropriate table, using the values in the [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) argument. If a column name is not in the [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) argument, you may want to provide a default value for it either in your provider code or in your database schema.

This method should return the content URI for the new row. To construct this, append the new row's \_ID (or other primary key) value to the table's content URI, using [withAppendedId()](https://developer.android.com/reference/android/content/ContentUris.html#withAppendedId(android.net.Uri, long)).

### Implementing the delete() method

The [delete()](https://developer.android.com/reference/android/content/ContentProvider.html#delete(android.net.Uri, java.lang.String, java.lang.String[])) method does not have to physically delete rows from your data storage. If you are using a sync adapter with your provider, you should consider marking a deleted row with a "delete" flag rather than removing the row entirely. The sync adapter can check for deleted rows and remove them from the server before deleting them from the provider.

### Implementing the update() method

The [update()](https://developer.android.com/reference/android/content/ContentProvider.html#update(android.net.Uri, android.content.ContentValues, java.lang.String, java.lang.String[])) method takes the same [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) argument used by [insert()](https://developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues)), and the same selection and selectionArgs arguments used by [delete()](https://developer.android.com/reference/android/content/ContentProvider.html#delete(android.net.Uri, java.lang.String, java.lang.String[])) and [ContentProvider.query()](https://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)). This may allow you to re-use code between these methods.

### Implementing the onCreate() method

The Android system calls [onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate()) when it starts up the provider. You should perform only fast-running initialization tasks in this method, and defer database creation and data loading until the provider actually receives a request for the data. If you do lengthy tasks in [onCreate()](https://developer.android.com/reference/android/content/ContentProvider.html#onCreate()), you will slow down your provider's startup. In turn, this will slow down the response from the provider to other applications.

The following two snippets demonstrate the interaction between [ContentProvider.onCreate()](https://developer.android.com/reference/android/content/ContentProvider#oncreate) and [Room.databaseBuilder()](https://developer.android.com/reference/android/arch/persistence/room/Room#databasebuilder). This snippet shows the implementation of [ContentProvider.onCreate()](https://developer.android.com/reference/android/content/ContentProvider#oncreate) where the database object is built and handles to the data access objects are created:

[Kotlin](https://developer.android.com/guide/topics/providers/content-provider-creating#kotlin)[Java](https://developer.android.com/guide/topics/providers/content-provider-creating#java)

// Defines the database name  
private const val DBNAME = "mydb"  
...  
class ExampleProvider : ContentProvider() {  
  
    // Defines a handle to the Room database  
    private lateinit var appDatabase: AppDatabase  
  
    // Defines a Data Access Object to perform the database operations  
    private var userDao: UserDao? = null  
  
    override fun onCreate(): Boolean {  
  
        // Creates a new database object.  
        appDatabase = Room.databaseBuilder(context, AppDatabase::class.java, DBNAME).build()  
  
        // Gets a Data Access Object to perform the database operations  
        userDao = appDatabase.userDao  
  
        return true  
    }  
  
    ...  
  
    // Implements the provider's insert method  
    override fun insert(uri: Uri, values: ContentValues?): Uri? {  
        // Insert code here to determine which DAO to use when inserting data, handle error conditions, etc.  
    }  
}

## Implementing ContentProvider MIME Types

The [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) class has two methods for returning MIME types:

[getType()](https://developer.android.com/reference/android/content/ContentProvider.html#getType(android.net.Uri))

One of the required methods that you must implement for any provider.

[getStreamTypes()](https://developer.android.com/reference/android/content/ContentProvider.html#getStreamTypes(android.net.Uri, java.lang.String))

A method that you're expected to implement if your provider offers files.

### MIME types for tables

The [getType()](https://developer.android.com/reference/android/content/ContentProvider.html#getType(android.net.Uri)) method returns a [String](https://developer.android.com/reference/java/lang/String.html) in MIME format that describes the type of data returned by the content URI argument. The [Uri](https://developer.android.com/reference/android/net/Uri.html) argument can be a pattern rather than a specific URI; in this case, you should return the type of data associated with content URIs that match the pattern.

For common types of data such as as text, HTML, or JPEG, [getType()](https://developer.android.com/reference/android/content/ContentProvider.html#getType(android.net.Uri)) should return the standard MIME type for that data. A full list of these standard types is available on the [IANA MIME Media Types](http://www.iana.org/assignments/media-types/index.htm) website.

For content URIs that point to a row or rows of table data, [getType()](https://developer.android.com/reference/android/content/ContentProvider.html#getType(android.net.Uri)) should return a MIME type in Android's vendor-specific MIME format:

* Type part: vnd
* Subtype part:
  + If the URI pattern is for a single row: android.cursor.**item**/
  + If the URI pattern is for more than one row: android.cursor.**dir**/
* Provider-specific part: vnd.<name>.<type>

You supply the <name> and <type>. The <name> value should be globally unique, and the <type> value should be unique to the corresponding URI pattern. A good choice for <name> is your company's name or some part of your application's Android package name. A good choice for the <type> is a string that identifies the table associated with the URI.

For example, if a provider's authority is com.example.app.provider, and it exposes a table named table1, the MIME type for multiple rows in table1 is:

vnd.android.cursor.**dir**/vnd.com.example.provider.table1

For a single row of table1, the MIME type is:

vnd.android.cursor.**item**/vnd.com.example.provider.table1

### MIME types for files

If your provider offers files, implement [getStreamTypes()](https://developer.android.com/reference/android/content/ContentProvider.html#getStreamTypes(android.net.Uri, java.lang.String)). The method returns a [String](https://developer.android.com/reference/java/lang/String.html) array of MIME types for the files your provider can return for a given content URI. You should filter the MIME types you offer by the MIME type filter argument, so that you return only those MIME types that the client wants to handle.

For example, consider a provider that offers photo images as files in .jpg, .png, and .gif format. If an application calls [ContentResolver.getStreamTypes()](https://developer.android.com/reference/android/content/ContentResolver.html#getStreamTypes(android.net.Uri, java.lang.String)) with the filter string image/\* (something that is an "image"), then the [ContentProvider.getStreamTypes()](https://developer.android.com/reference/android/content/ContentProvider.html#getStreamTypes(android.net.Uri, java.lang.String)) method should return the array:

{ "image/jpeg", "image/png", "image/gif"}

If the app is only interested in .jpg files, then it can call [ContentResolver.getStreamTypes()](https://developer.android.com/reference/android/content/ContentResolver.html#getStreamTypes(android.net.Uri, java.lang.String)) with the filter string \*\/jpeg, and [ContentProvider.getStreamTypes()](https://developer.android.com/reference/android/content/ContentProvider.html#getStreamTypes(android.net.Uri, java.lang.String)) should return:

{"image/jpeg"}

If your provider doesn't offer any of the MIME types requested in the filter string, [getStreamTypes()](https://developer.android.com/reference/android/content/ContentProvider.html#getStreamTypes(android.net.Uri, java.lang.String)) should return null.

## Implementing a contract class

A contract class is a public final class that contains constant definitions for the URIs, column names, MIME types, and other meta-data that pertain to the provider. The class establishes a contract between the provider and other applications by ensuring that the provider can be correctly accessed even if there are changes to the actual values of URIs, column names, and so forth.

A contract class also helps developers because it usually has mnemonic names for its constants, so developers are less likely to use incorrect values for column names or URIs. Since it's a class, it can contain Javadoc documentation. Integrated development environments such as Android Studio can auto-complete constant names from the contract class and display Javadoc for the constants.

Developers can't access the contract class's class file from your application, but they can statically compile it into their application from a .jar file you provide.

The [ContactsContract](https://developer.android.com/reference/android/provider/ContactsContract.html) class and its nested classes are examples of contract classes.

## Implementing content provider permissions

Permissions and access for all aspects of the Android system are described in detail in the topic [Security and permissions](https://developer.android.com/guide/topics/security/security.html). The topic [Data storage](https://developer.android.com/guide/topics/data/data-storage.html) also described the security and permissions in effect for various types of storage. In brief, the important points are:

* By default, data files stored on the device's internal storage are private to your application and provider.
* [SQLiteDatabase](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html) databases you create are private to your application and provider.
* By default, data files that you save to external storage are public and world-readable. You can't use a content provider to restrict access to files in external storage, because other applications can use other API calls to read and write them.
* The method calls for opening or creating files or SQLite databases on your device's internal storage can potentially give both read and write access to all other applications. If you use an internal file or database as your provider's repository, and you give it "world-readable" or "world-writeable" access, the permissions you set for your provider in its manifest won't protect your data. The default access for files and databases in internal storage is "private", and for your provider's repository you shouldn't change this.

If you want to use content provider permissions to control access to your data, then you should store your data in internal files, SQLite databases, or the "cloud" (for example, on a remote server), and you should keep files and databases private to your application.

### Implementing permissions

All applications can read from or write to your provider, even if the underlying data is private, because by default your provider does not have permissions set. To change this, set permissions for your provider in your manifest file, using attributes or child elements of the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. You can set permissions that apply to the entire provider, or to certain tables, or even to certain records, or all three.

You define permissions for your provider with one or more [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) elements in your manifest file. To make the permission unique to your provider, use Java-style scoping for the [android:name](https://developer.android.com/guide/topics/manifest/permission-element.html#nm) attribute. For example, name the read permission com.example.app.provider.permission.READ\_PROVIDER.

The following list describes the scope of provider permissions, starting with the permissions that apply to the entire provider and then becoming more fine-grained. More fine-grained permissions take precedence over ones with larger scope:

Single read-write provider-level permission

One permission that controls both read and write access to the entire provider, specified with the [android:permission](https://developer.android.com/guide/topics/manifest/provider-element.html#prmsn) attribute of the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element.

Separate read and write provider-level permission

A read permission and a write permission for the entire provider. You specify them with the [android:readPermission](https://developer.android.com/guide/topics/manifest/provider-element.html#rprmsn) and [android:writePermission](https://developer.android.com/guide/topics/manifest/provider-element.html#wprmsn) attributes of the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. They take precedence over the permission required by [android:permission](https://developer.android.com/guide/topics/manifest/provider-element.html#prmsn).

Path-level permission

Read, write, or read/write permission for a content URI in your provider. You specify each URI you want to control with a [<path-permission>](https://developer.android.com/guide/topics/manifest/path-permission-element.html) child element of the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. For each content URI you specify, you can specify a read/write permission, a read permission, or a write permission, or all three. The read and write permissions take precedence over the read/write permission. Also, path-level permission takes precedence over provider-level permissions.

Temporary permission

A permission level that grants temporary access to an application, even if the application doesn't have the permissions that are normally required. The temporary access feature reduces the number of permissions an application has to request in its manifest. When you turn on temporary permissions, the only applications that need "permanent" permissions for your provider are ones that continually access all your data.

Consider the permissions you need to implement an email provider and app, when you want to allow an outside image viewer application to display photo attachments from your provider. To give the image viewer the necessary access without requiring permissions, set up temporary permissions for content URIs for photos. Design your email app so that when the user wants to display a photo, the app sends an intent containing the photo's content URI and permission flags to the image viewer. The image viewer can then query your email provider to retrieve the photo, even though the viewer doesn't have the normal read permission for your provider.

To turn on temporary permissions, either set the [android:grantUriPermissions](https://developer.android.com/guide/topics/manifest/provider-element.html#gprmsn) attribute of the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element, or add one or more [<grant-uri-permission>](https://developer.android.com/guide/topics/manifest/grant-uri-permission-element.html) child elements to your [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. If you use temporary permissions, you have to call [Context.revokeUriPermission()](https://developer.android.com/reference/android/content/Context.html#revokeUriPermission(android.net.Uri, int)) whenever you remove support for a content URI from your provider, and the content URI is associated with a temporary permission.

The attribute's value determines how much of your provider is made accessible. If the attribute is set to true, then the system will grant temporary permission to your entire provider, overriding any other permissions that are required by your provider-level or path-level permissions.

If this flag is set to false, then you must add [<grant-uri-permission>](https://developer.android.com/guide/topics/manifest/grant-uri-permission-element.html) child elements to your [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. Each child element specifies the content URI or URIs for which temporary access is granted.

To delegate temporary access to an application, an intent must contain the [FLAG\_GRANT\_READ\_URI\_PERMISSION](https://developer.android.com/reference/android/content/Intent.html#FLAG_GRANT_READ_URI_PERMISSION) or the [FLAG\_GRANT\_WRITE\_URI\_PERMISSION](https://developer.android.com/reference/android/content/Intent.html#FLAG_GRANT_WRITE_URI_PERMISSION) flags, or both. These are set with the [setFlags()](https://developer.android.com/reference/android/content/Intent.html#setFlags(int)) method.

If the [android:grantUriPermissions](https://developer.android.com/guide/topics/manifest/provider-element.html#gprmsn) attribute is not present, it's assumed to be false.

## The <provider> element

Like [Activity](https://developer.android.com/reference/android/app/Activity.html) and [Service](https://developer.android.com/reference/android/app/Service.html) components, a subclass of [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) must be defined in the manifest file for its application, using the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element. The Android system gets the following information from the element:

Authority ([android:authorities](https://developer.android.com/guide/topics/manifest/provider-element.html#auth))

Symbolic names that identify the entire provider within the system. This attribute is described in more detail in the section [Designing Content URIs](https://developer.android.com/guide/topics/providers/content-provider-creating#ContentURI).

Provider class name ( [android:name](https://developer.android.com/guide/topics/manifest/provider-element.html#nm) )

The class that implements [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html). This class is described in more detail in the section [Implementing the ContentProvider class](https://developer.android.com/guide/topics/providers/content-provider-creating#ContentProvider).

Permissions

Attributes that specify the permissions that other applications must have in order to access the provider's data:

* [android:grantUriPermssions](https://developer.android.com/guide/topics/manifest/provider-element.html#gprmsn): Temporary permission flag.
* [android:permission](https://developer.android.com/guide/topics/manifest/provider-element.html#prmsn): Single provider-wide read/write permission.
* [android:readPermission](https://developer.android.com/guide/topics/manifest/provider-element.html#rprmsn): Provider-wide read permission.
* [android:writePermission](https://developer.android.com/guide/topics/manifest/provider-element.html#wprmsn): Provider-wide write permission.

Permissions and their corresponding attributes are described in more detail in the section [Implementing Content provider permissions](https://developer.android.com/guide/topics/providers/content-provider-creating#Permissions).

Startup and control attributes

These attributes determine how and when the Android system starts the provider, the process characteristics of the provider, and other run-time settings:

* [android:enabled](https://developer.android.com/guide/topics/manifest/provider-element.html#enabled): Flag allowing the system to start the provider.
* [android:exported](https://developer.android.com/guide/topics/manifest/provider-element.html#exported): Flag allowing other applications to use this provider.
* [android:initOrder](https://developer.android.com/guide/topics/manifest/provider-element.html#init): The order in which this provider should be started, relative to other providers in the same process.
* [android:multiProcess](https://developer.android.com/guide/topics/manifest/provider-element.html#multi): Flag allowing the system to start the provider in the same process as the calling client.
* [android:process](https://developer.android.com/guide/topics/manifest/provider-element.html#proc): The name of the process in which the provider should run.
* [android:syncable](https://developer.android.com/guide/topics/manifest/provider-element.html#sync): Flag indicating that the provider's data is to be sync'ed with data on a server.

The attributes are fully documented in the dev guide topic for the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element.

Informational attributes

An optional icon and label for the provider:

* [android:icon](https://developer.android.com/guide/topics/manifest/provider-element.html#icon): A drawable resource containing an icon for the provider. The icon appears next to the provider's label in the list of apps in Settings > Apps > All.
* [android:label](https://developer.android.com/guide/topics/manifest/provider-element.html#label): An informational label describing the provider or its data, or both. The label appears in the list of apps in Settings > Apps > All.

The attributes are fully documented in the dev guide topic for the [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html) element.

## Intents and data access

Applications can access a content provider indirectly with an [Intent](https://developer.android.com/reference/android/content/Intent.html). The application does not call any of the methods of [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) or [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html). Instead, it sends an intent that starts an activity, which is often part of the provider's own application. The destination activity is in charge of retrieving and displaying the data in its UI. Depending on the action in the intent, the destination activity may also prompt the user to make modifications to the provider's data. An intent may also contain "extras" data that the destination activity displays in the UI; the user then has the option of changing this data before using it to modify the data in the provider.

You may want to use intent access to help ensure data integrity. Your provider may depend on having data inserted, updated, and deleted according to strictly defined business logic. If this is the case, allowing other applications to directly modify your data may lead to invalid data. If you want developers to use intent access, be sure to document it thoroughly. Explain to them why intent access using your own application's UI is better than trying to modify the data with their code.

Handling an incoming intent that wishes to modify your provider's data is no different from handling other intents. You can learn more about using intents by reading the topic [Intents and Intent Filters](https://developer.android.com/guide/components/intents-filters.html).

For sample code related to this page, refer to the [Note Pad sample application.](https://developer.android.com/resources/samples/NotePad/index.html)

For additional related information, refer to [Calendar Provider.](https://developer.android.com/guide/topics/providers/calendar-provider.html)