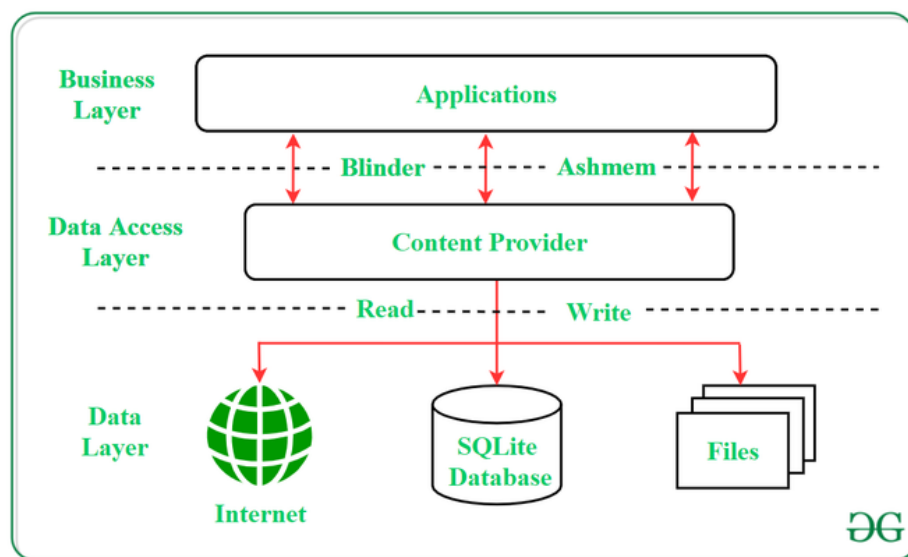




Content Providers in Android with Example

[Read](#)[Courses](#)[Practice](#)

In [Android](#), Content Providers are a very important [component](#) that serves the purpose of a relational database to store the data of applications. The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements. Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in [SQLite Database](#), in files, or even on a network. In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.



Content URI

Content URI (Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

Structure of a Content URI:

content://authority/optionalPath/optionalID

Details of different parts of Content URI:

- **content://** – Mandatory part of the URI as it represents that the given URI is a Content URI.
- **authority** – Signifies the name of the content provider like contacts, browser, etc. This part must be unique for every content provider.
- **optionalPath** – Specifies the type of data provided by the content provider. It is essential as this part helps content providers to support different types of data that are not related to each other like audio and video files.
- **optionalID** – It is a numeric value that is used when there is a need to access a particular record.

If an ID is mentioned in a URI then it is an id-based URI otherwise a directory-based URI.

Operations in Content Provider

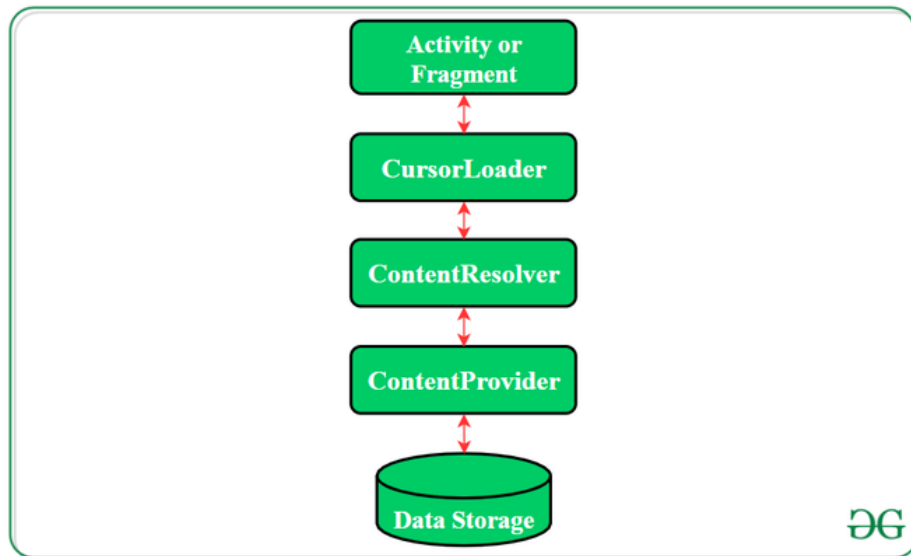
Four fundamental operations are possible in Content Provider namely **Create**, **Read**, **Update**, and **Delete**. These operations are often termed as **CRUD operations**.

- **Create:** Operation to create data in a content provider.
- **Read:** Used to fetch data from a content provider.
- **Update:** To modify existing data.
- **Delete:** To remove existing data from the storage.

Working of the Content Provider

UI components of android applications like [Activity](#) and [Fragments](#) use an object **CursorLoader** to send query requests to **ContentResolver**. The **ContentResolver** object sends requests (like create, read, update, and delete) to the **ContentProvider** as a client. After receiving a request,

ContentProvider process it and returns the desired result. Below is a diagram to represent these processes in pictorial form.



Creating a Content Provider

Following are the steps which are essential to follow in order to create a Content Provider:

- Create a class in the same directory where the that **MainActivity** file resides and this class must extend the ContentProvider base class.
- To access the content, define a content provider URI address.
- Create a database to store the application data.
- Implement the **six abstract methods** of ContentProvider class.
- Register the content provider in **AndroidManifest.xml** file using **<provider>** tag.

Following are the six abstract methods and their description which are essential to override as the part of ContentProvider class:

| Abstract Method | Description |
|-----------------|--|
| query() | A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object. |

| Abstract Method | Description |
|-----------------|---|
| insert() | To insert a new row in the database of the content provider. It returns the content URI of the inserted row. |
| update() | This method is used to update the fields of an existing row. It returns the number of rows updated. |
| delete() | This method is used to delete the existing rows. It returns the number of rows deleted. |
| getType() | This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI. |
| onCreate() | As the content provider is created, the android system calls this method immediately to initialise the provider. |

Example

The prime purpose of a content provider is to serve as a central repository of data where users can store and can fetch the data. The access of this repository is given to other applications also but in a safe manner in order to serve the different requirements of the user. The following are the steps involved in implementing a content provider. In this content provider, the user can store the name of persons and can fetch the stored data. Moreover, another application can also access the stored data and can display the data.

Note: Following steps are performed on Android Studio version 4.0

Creating a Content Provider:

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Select language as Java/Kotlin.
3. Choose empty activity as a template
4. Select the minimum SDK as per your need.

Step 2: Modify the strings.xml file

All the strings used in the activity are stored here.

XML

```
<resources>
    <string name="app_name">Content_Provider_In_Android</string>
    <string name="hintText">Enter User Name</string>
    <string name="heading">Content Provider In Android</string>
    <string name="insertButtonText">Insert Data</string>
    <string name="loadButtonText">Load Data</string>
</resources>
```

Step 3: Creating the Content Provider class

1. Click on File, then New => Other => ContentProvider.
2. Name the ContentProvider
3. Define authority (it can be anything for example
 “com.demo.user.provider”)
4. Select **Exported** and **Enabled** option
5. Choose the language as Java/Kotlin

This class extends the ContentProvider base class and override the six abstract methods. Below is the complete code to define a content provider.

Java

```
package com.example.contentprovidersinandroid;

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;

public class MyContentProvider extends ContentProvider {
    public MyContentProvider() {
    }

    // defining authority so that other application can access it
    static final String PROVIDER_NAME = "com.demo.user.provider";

    // defining content URI
    static final String URL = "content://" + PROVIDER_NAME + "/users";

    // parsing the content URI
    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 {
        // to match the content URI
        // every time user access table under content provider
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);


        // to access whole table
        uriMatcher.addURI(PROVIDER_NAME, "users", uriCode);

        // to access a particular row
        // of the table
        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);
        }
    }
}

```

// creating the database

90% Refund @Courses Java Arrays Java Strings Java OOPs Java Collection Java 8 Tutorial Java Mul

 Related Articles →

```

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;
    }

    // adding data to the database
    @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;
}

// creating object of database
// to perform query
private SQLiteDatabase db;

// declaring name of the database
static final String DATABASE_NAME = "UserDB";

// declaring table name of the database
static final String TABLE_NAME = "Users";

// declaring version of the database
static final int DATABASE_VERSION = 1;

// sql query to create the table
static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME
    + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
    + " name TEXT NOT NULL);";

// creating a database
private static class DatabaseHelper extends SQLiteOpenHelper {

    // defining a constructor
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // creating a table in the database
    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersio

        // sql query to drop a table
        // having similar name

```