ContentProvider and UI Demonstration exercises.

1. App Skeleton creation

Step1: create and run emulator (demonstrated by trainer).

Step 2: Create single activity application named “ContentProviderSample” with package “com.training” (demonstrated by trainer).

Step 3: Create Run configuration and run the app (demonstrated by trainer).

1. Data Base creation

Create data base file add fill it with some test data.

See step by step instruction with comments below (ask trainer if you need assistance):

# go to platform-tools folder in sdk folder (change the path to appropriate value)

cd /android/sdk/platform-tools/

# use adb to connect to localhost

./adb connect localhost

# check the list of devices

./adb devices

# run shell on emulator

./adb -s localhost:5555 shell

# go to application folder

cd /data/data/com.training.contentprovidersamle

# create new folder where we will keep database files

mkdir databases

# go inside

cd databases

# use sqlite3 tool to create database

sqlite3 books

# create table

CREATE TABLE books (\_id INT NOT\_NULL AUTO\_INCREMENT PRIMARY\_KEY, title VARCHAR(100), author VARCHAR(100), genre VARCHAR(50));

# check it's empty

SELECT \* FROM books;

# fill it with some test data

# first book

INSERT INTO books (title, author, genre) VALUES ('[Android Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides](http://www.amazon.com/Android-Programming-Ranch-Guide-Guides/dp/0321804333/ref=sr_1_1?s=books&ie=UTF8&qid=1378470637&sr=1-1&keywords=Android))', '[Bill Phillips](http://www.amazon.com/Bill-Phillips/e/B00C9F7BHE/ref=sr_ntt_srch_lnk_1?qid=1378470637&sr=1-1) and [Brian Hardy](http://www.amazon.com/Brian-Hardy/e/B00C9F5OTQ/ref=sr_ntt_srch_lnk_1?qid=1378470637&sr=1-1)', 'programming');

# second one

INSERT INTO books (title, author, genre) VALUES ('[Android Programming Painless (Tutorial Book)](http://www.amazon.com/Android-Programming-Painless-Tutorial-ebook/dp/B0099QCVU8/ref=sr_1_2?s=books&ie=UTF8&qid=1378470637&sr=1-2&keywords=Android) ', '[Camilus Raynaldo](http://www.amazon.com/Camilus-Raynaldo/e/B008MZUYBU/ref=sr_ntt_srch_lnk_2?qid=1378470637&sr=1-2)', 'programming');

# one more

INSERT INTO books (title, author, genre) VALUES ('[Professional Android 4 Application Development](http://www.amazon.com/Professional-Android-4-Application-Development/dp/1118102274/ref=sr_1_3?s=books&ie=UTF8&qid=1378470637&sr=1-3&keywords=Android)', '[Reto Meier](http://www.amazon.com/Reto-Meier/e/B002BMF4OC/ref=sr_ntt_srch_lnk_3?qid=1378470637&sr=1-3)', 'programming');

# and finally

INSERT INTO books (title, author, genre) VALUES ('[Android Tablets For Dummies (For Dummies (Computer/Tech))](http://www.amazon.com/Android-Tablets-Dummies-Computer-Tech/dp/111854319X/ref=sr_1_5?s=books&ie=UTF8&qid=1378470637&sr=1-5&keywords=Android)', '[Dan Gookin](http://www.amazon.com/Dan-Gookin/e/B000APZGIG/ref=sr_ntt_srch_lnk_5?qid=1378470637&sr=1-5)', 'programming');

# check the result

SELECT \* FROM books;

# exit the sqlite3 tool

.exit

# check the db file is created now

ls

1. Create Content Provider class and registry it

Step 1: Copy from sample project provided by trainer or create new class in src/com.training.contentprovidersample folder new class with name BookContentProvider.java that extands android.content.ContentProvider

Step2: registry this class as content provider in AndroidManifest.xml file

To do so, add the following lines after <activity> tag:

<provider android:name="com.training.contentprovidersample.BooksContentProvider"

android:authorities="books">

</provider>

1. Content Provider members initialization

In BooksContentProvider.java file add the following lines at the beginning. It contains some members we will use in our provider logic. See the comments inline:

**public class BooksContentProvider extends ContentProvider {**

/\*\* Reference to SQLiteOpenHelper.

\* We will use our own private Database Helper class

\* to track the db access and updates (see this class definition below)

\*/

**private** DatabaseHelper mDatabaseHelper;

/\*\*

\* this is the name of our db file that we created on previous step

\*/

**private** **static** **final** String *DATABASE\_NAME* = "books.db";

/\*\* Database version. \*/

**private** **static** **final** **int** *DATABASE\_VERSION* = 1;

/\*\*

\* Constant mapped to uri "com.training.contentprovidersample/books"} via mUriMatcher.

\* We will use this constant in switch to decide what query we should perform depends on requested URI. You will see it later.

\*/

**private** **static** **final** **int** *BOOKS* = 1;

/\*\*

\* Constant mapped to uri "com.training.contentprovidersample/books/#" via mUriMatcher.

\* We will use this constant in switch to decide what query we should perform depends on requested URI. You will see it later.

\*/

**private** **static** **final** **int** *BOOKS\_ID* = 2;

/\*\* Local UriMatcher.

\* this class will help us parse the passed URI to content provider

\*/

**private** **static** **final** UriMatcher *mUriMatcher*;

/\*\* Now we initialize UriMatcher and add two URI that we support in our content provider

\* first parameter is authority, second parameter is path and the last parameter is code that will be assigned for the URI if it matches the path

\* We support two URI: first one is to get all books, and the second one to get particular book by id

\*/

**static** {

*mUriMatcher* = **new** UriMatcher(UriMatcher.*NO\_MATCH*);

*mUriMatcher*.addURI("com.training.contentprovidersample", "books", *BOOKS*);

*mUriMatcher*.addURI("com.training.contentprovidersample", "books/#", *BOOKS\_ID*);

}

1. DatabaseHelper implementation

Now let’s implement our own DatabaseHelper that in our case extends SQLiteOpenHelper

Add the following code right after code from step 4:

/\*\* Utility class for db creation and update. \*/

**private** **static** **class** DatabaseHelper **extends** SQLiteOpenHelper {

/\*\*

\* Default constructor.

\* **@param** context - parent context.

\* We invoke super class constructor here and pass database name and version

\* Third parameter is CursorFactory, we don't need to customize it, so it's null

\*/

DatabaseHelper(Context context) {

**super**(context, *DATABASE\_NAME*, **null**, *DATABASE\_VERSION*);

}

/\*\*

\* Because we already created table manually in our db on previous steps,

\* we don't want to recreate it again.

\* But in real application you might want to create table in a code.

\* Here is how you can do it

\*/

@Override

**public** **void** onCreate(SQLiteDatabase db) {

/\*

db.execSQL("Create table books"

+ "( \_id INTEGER PRIMARY KEY AUTOINCREMENT, "

+ " title VARCHAR(100), "

+ " author VARCHAR(100), "

+ " genre VARCHAR(50));");

\*/

}

/\*\*

\* In case your db version is updated, you might want to make some update in db structure.

\* In this simple app we don't want to do any changes.

\* Still the commented code has some stupid implementation: drop all tables and recreate them.

\*/

@Override

**public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {

// drop db

/\*

db.execSQL("DROP TABLE IF EXISTS books");

// create new db

onCreate(db);

\*/

}

}

1. Init our content provider

Now we are ready to implement our Content Provider logic. We will start from implementation of onCreate method.

If wizard has created this method for you, all you need is just fill it with functionality as shown below, otherwise add it:

/\*\*

\* This method is initial point for Content Provider.

\* It's invoked even before any client tries get access to provider.

\* This is good place to do some initialization.

\*/

@Override

**public** **boolean** onCreate() {

/\*\*

\* All we want is create our internal Database Helper here

\*/

mDatabaseHelper = **new** DatabaseHelper(getContext());

**boolean** result = **false**;

**if** (mDatabaseHelper != **null**) {

// success so return true, we are ready to handle requests from clients

result = **true**;

}

// something wrong, so return false

**return** result;

}

1. Handle query request in provider

Now we are ready to do some real job and we will start with query method. If wizard has created query method for you, all you need is just fill it with functionality as shown below, otherwise add it:

/\*\*

\* In our sample application we will handle two types of URI requests:

\* 1: "com.training.contentprovidersample/books" – to return whole list of books

\* 2: "com.training.contentprovidersample/books/#” – to return specific book details that matches the book id

\* (for instance com.training.contentprovidersample/books/2 to get book with id equals 2)

\* uri parameter - is URI itself that we will parse using our uri matcher

\* projection - is the list of columns that expected to be in result cursor

\* selection - is selection criteria for query

\* selectionArgs - the list of values that used in selection criteria

\* sortOrder - sorting order that client want to get for result data

\*/

@Override

**public** Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

/\*

\* Choose the table to query and a sort order based on the code returned for the incoming URI.

\* We use our uri matcher here

\*/

**switch** (*mUriMatcher*.match(uri)) {

// If the incoming URI was for all of table3

**case** *BOOKS*:

// let's sort books by title by default

**if** (TextUtils.*isEmpty*(sortOrder)) sortOrder = "title ASC";

**break**;

// If the incoming URI was for a single row

**case** *BOOKS\_ID*:

/\*

\* 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

\*/

selection = selection + "\_ID = " + uri.getLastPathSegment();

**break**;

**default**:

// If the URI is not recognized, you should do some error handling here.

}

// the code to actually do the query

// initialize the query builder

SQLiteQueryBuilder qb = **new** SQLiteQueryBuilder();

// get access to database via our helper

SQLiteDatabase db = mDatabaseHelper.getReadableDatabase();

// set the appropriate table we want to select data from (we have only one table - books)

qb.setTables("books");

// now execute query and get result cursor

Cursor c = qb.query(db, projection, selection, selectionArgs, **null**, **null**, sortOrder);

**return** c;

}

1. To summarize

We are done with Content Provider implementation. You can overwrite and implement insert/delete/update method the same way. We will not do it in this sample. Let’s consider it as your homework.

1. Invoke Content Provider

Not let’s test our provider. In your MainActivty.java in onCreate method after setContentView(…) add the following code:

// projection specifies the columns we want to get in result data.

String[] projection = {"\_id", "title", "author", "genre"};

/\*\*

\* The most important part of Content Provider invocation is URI

\* shema = content://

\* authority = com.training.contentprovidersample (remember we registered it in Manifest file?)

\* path = books (that our content provider can parse. Remember we use UriMatcher to parse it?)

\*/

Uri mBooks = Uri.*parse*("content://com.training.contentprovidersample/books");

/\*\*

\* Now invoke our content provider and get Cursor as result of operation

\* getContentResolver used to get instance of ContentResolved that is responsible for

\* finding appropriate ContentProvider basing on authority

\* First parameter - URI

\* Second parameter - projection

\* Thirst parameter - selection if we want specify some selection (we don't now).

\* It might be "title = ?" for example

\* Fourth parameter - selection parameters (all '?' characters from selection criteria

\* will be replaced by element of array from this parameter). For example {"Android Book Title"}

\* And last parameter is sorting order. For instance "title ASC" to sort by title

\*/

Cursor queryCursor = getContentResolver().query(mBooks, projection, "", **null**, **null**);

// check the cursor on validity

**if** (queryCursor == **null**) {

Log.*d*("SampleContentProvider", "can't create cursor");

} **else** {

// Iterate through all cursor elements to get all data from our query

**while** (queryCursor.moveToNext()) {

// each cursor element has data according to projection

// Use type specific method to get appropriate column data

**int** id = queryCursor.getInt(0);

String title = queryCursor.getString(1);

String author = queryCursor.getString(2);

String genre = queryCursor.getString(3);

// Log it to LogCat console

Log.*d*("SampleContentProvider", "Book[" + id + "] = title: " + title

+ " author: " + author + " genre: " + genre);

}

// When we are done, close cursor

queryCursor.close();

}

1. Run the app and check the results in LogCat

List of books from db sorted by title should appear in LogCat console when application is launched.

1. Main layout UI creation

Now when we have ContentProvider working, let’s implement UI sample to display the list of UI.

First we need to update our activity\_main.xml document (that is used as default layout for our MainActivity).

If you already have TextView for “Hello World” just change the android:text field of this tag to "@string/table\_title", otherwise add TextView tag inside RelativeLayout tag. As result you should have:

<TextView

android:id=*"@+id/textView1"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"@string/table\_title"* />

Below this tag add the ListView as shown below:

<ListView

android:id=*"@+id/listView1"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*

android:layout\_alignParentLeft=*"true"*

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

android:layout\_marginTop=*"10dp"/*>

Finally add the string resource for table title. To do so go to “value” folder in projectview and open “strings.xml”. Add new string resource with name “table\_title” and value “Android books”

Review the result in graphical view of activity\_main.xml file. You should have TextView on top and the ListView below it.

1. List item layout UI

Now we need to create UI layout for the row in our list. Each list item should have TextView aligned on top to display book title value. Below we should have two TextView to display author and genre values. Author should be aligned left and genre should be aligned right. To get this implemented better to use combination Linear and Relative layout. Create new Android XML file in layout folder of the project and fill it with the following content:

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

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

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

android:orientation=*"vertical"* >

<TextView

android:id=*"@+id/title"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*

android:textAppearance=*"?android:attr/textAppearanceLarge"* />

<RelativeLayout

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*>

<TextView

android:id=*"@+id/author"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_alignParentLeft=*"true"*

android:textAppearance=*"?android:attr/textAppearanceMedium"* />

<TextView

android:id=*"@+id/genge"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_alignParentRight=*"true"*

android:layout\_alignParentTop=*"true"*

android:textAppearance=*"?android:attr/textAppearanceMedium"* />

</RelativeLayout>

</LinearLayout>

1. Join the model and UI

Now when we have Content Provider that provides us with data about books and we have UI to display this data it’s time to join them. To do so we will use Adapter. And Android already has suitable adapter for us. It’s SimpleCursorAdapter, that uses the data from Cursor to display it using ListView UI widget.

In MainActivity.java file in onCreate method remove all code after line: Cursor queryCursor = getContentResolver().query(mBooks, projection, "", **null**, **null**);

We don’t want to log the data we want to display it now. Add the following code instead of removed one:

/\*\*

\* This string array specifies all columns that will be used from Cursor.

\* All values of these columns will be added to appropriate TextView

\*/

String[] fromColumns = {"title", "author", "genre"};

/\*\*

\* Here we specify the IDs of TextView widgets that will be used to display each specific column.

\* They should correspond to previous array with column name

\*/

**int**[] toViews = {R.id.*title*, R.id.*author*, R.id.*genge*};

/\*\*

\* Now we are ready to create SimpleCursorAdapter.

\* first parameter is context

\* second parameter is layout that will be used for row representation in ListView

\* third parameter is cursor with data

\* fourth parameter is array of columns from cursor

\* fifth parameter is array of IDs for UI widgets

\* the last one is flags parameter that tells adapter how to load data (see documentation for details).

\*/

SimpleCursorAdapter adapter = **new** SimpleCursorAdapter(**this**, R.layout.*row\_layout*, queryCursor, fromColumns, toViews, 0);

/\*\*

\* We need get the link to ListView class from our UI.

\* To do so we use findViewById method with UI widget id as parameter

\*/

ListView listView = (ListView)findViewById(R.id.*listView1*);

/\*\*

\* Now join the ListView and adapter for it

\*/

listView.setAdapter(adapter);

1. Run the application and observe the results

Please ask your trainer if you have any questions!!!

1. HomeWork

Try to enhance this sample to add UI and logic to get ability to insert new books to the database.