# Content Provider

Session 7

# Objective



- Content provider
- Android’s Built-in Providers
- Architecture of Content Providers
- Creating content provider
- Exposing Access to the Data Source
- Using content providers
- Using the Cursor
- Querying for Content
- Adding, Updating, and Deleting Content

# Content Provider



- A content provider is a wrapper around data.
- Android allows you to expose data sources (or data providers) through a REST (Representational State Transfer)
- To retrieve data from a content provider or save data into a content provider, you will need to use a set of REST-like URIs.

- For example:

content://com.android.book.BookProvider/books

# Content Provider



- For internal data access, an application can use any data storage/ access mechanism that it deems suitable, such as the following:
  - Preferences: A set of key/value pairs that you can persist to store application preferences
  - Files: Files internal to applications, which you can store on a removable storage medium
  - SQLite: SQLite databases, each of which is private to the package that creates that database
  - Network: A mechanism that lets you retrieve or store data externally through the Internet

# Android’s Built-in Providers



- Android comes with a number of built-in content providers:
  - Browser
  - CallLog
  - Contacts
    - People, Phones, Photos, Groups
  - MediaStore
    - Audio
      - Albums, Artists, Genres, Playlists
    - Images
    - Thumbnails
    - Video
  - Settings

# Architecture of Content Providers



- Overall, the content-provider approach has parallels to the following industry abstractions:
  - Web sites
  - REST
  - Web services
  - Stored procedures
- This authority registration occurs in the AndroidManifest.xml.
- Register providers in AndroidManifest.xml:

```
<provider android:name="SomeProvider"
    android:authorities="com.your-company.SomeProvider" />
<provider android:name="NotePadProvider"
    android:authorities="com.google.provider.NotePad"/>
```

# Structure of Android Content URIs



- Content URIs in Android look similar to HTTP URIs, except that they start with content and have this general form:

content://*/*/*
or
content://authority-name/path-segment1/path-segment2/etc

- Here's an example URI that identifies a note numbered 23 in a database of notes:

content://com.google.provider.NotePad/notes/23

The notes and 23 portions of the path section are called path segments.

# Creating Content Provider



- Sample skeleton code for a new Content Provider is shown below:

```
public class MyProvider extends ContentProvider {
    @Override
    public boolean onCreate() {
        // TODO Construct the underlying database.
        return true;
    }
}
```

- Create and configure a Uri Matcher to parse URIs and determine their forms:

```
public class MyProvider extends ContentProvider {
    private static final String myURI = "content://vn.vtc.myapp/items";
    public static final Uri CONTENT_URI = Uri.parse(myURI);
    @Override
    public boolean onCreate() {
        // TODO: Construct the underlying database.
        return true;
    }
    ...
}
```

- Expose queries and transactions on your Content Provider by implementing the delete, insert, update, and query methods.
- These methods are the interface used by the Content Resolver to access the underlying data.
- The skeleton code for implementing queries and transactions within a Content Provider:

```
....
@Override
public Uri insert(Uri _uri, ContentValues _initialValues) {
}

@Override
public int delete(Uri uri, String where, String[] whereArgs) {
}
...
```

# Using the Cursor



- A cursor is a collection of rows.
- You need to use moveToFirst() because the cursor is positioned before the first row.
- You need to know the column names.
- You need to know the column types.
- All field-access methods are based on column number, so you must convert the column name to a column number first.
- The cursor is a random cursor (you can move forward and backward and you can jump).
- Because the cursor is a random cursor, you can ask it for a row count.

# Using the Cursor



- To help you learn where the cursor is, Android provides the following methods:
  - isBeforeFirst()
  - isAfterLast()
  - isClosed()

- Using these methods, you can also use a for loop to navigate through the cursor instead of the while loop

```
for (cur.moveToFirst(); !cur.isAfterLast(); cur.moveToNext()) {
    int nameColumn = cur.getColumnIndex(People.NAME);
    int phoneColumn = cur.getColumnIndex(People.NUMBER);
    String name = cur.getString(nameColumn);
    String phoneNumber = cur.getString(phoneColumn);
}
```

# Querying for Content



- Using the query method on the ContentResolver object, pass in:
  - The URI of the Content Provider data you want to query.
  - A projection that lists the columns you want to include in the result set.
  - A where clause that defines the rows to be returned. You can include ? wildcards that will be replaced by the values passed into the selection argument parameter.
  - An array of selection argument strings that will replace the ?s in the where clause.
  - A string that describes the order of the returned rows.

# Querying for Content



- The snippset below shows how to use a Content Resolver to apply a query to a Content Provider:

```
ContentResolver cr = getContentResolver();
// Return all rows
Cursor allRows = cr.query(MyProvider.CONTENT_URI, null, null, null, null);
// Return all columns for rows where column 3 equals a set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows = cr.query(MyProvider.CONTENT_URI, null, where, null, order);
```

# Insert Content



- The simple insert method will return a URI to the newly added record, while bulkInsert returns the number of successfully added rows:

```
// Get the Content Resolver
ContentResolver cr = getContentResolver();
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
Uri myRowUri = cr.insert(MyProvider.CONTENT_URI, newValues);
// Create a new row of values to insert.
ContentValues[] valueArray = new ContentValues[5];
// TODO: Create an array of new rows
int count = cr.bulkInsert(MyProvider.CONTENT_URI, valueArray);
```

# Delete Content



- Call delete on the Content Resolver, passing in the URI of the row you want to remove:

```
ContentResolver cr = getContentResolver();
// Remove a specific row.
cr.delete(myRowUri, null, null);
// Remove the first five rows.
String where = "_id < 5";
cr.delete(MyProvider.CONTENT_URI, where, null);
```

# Update Content



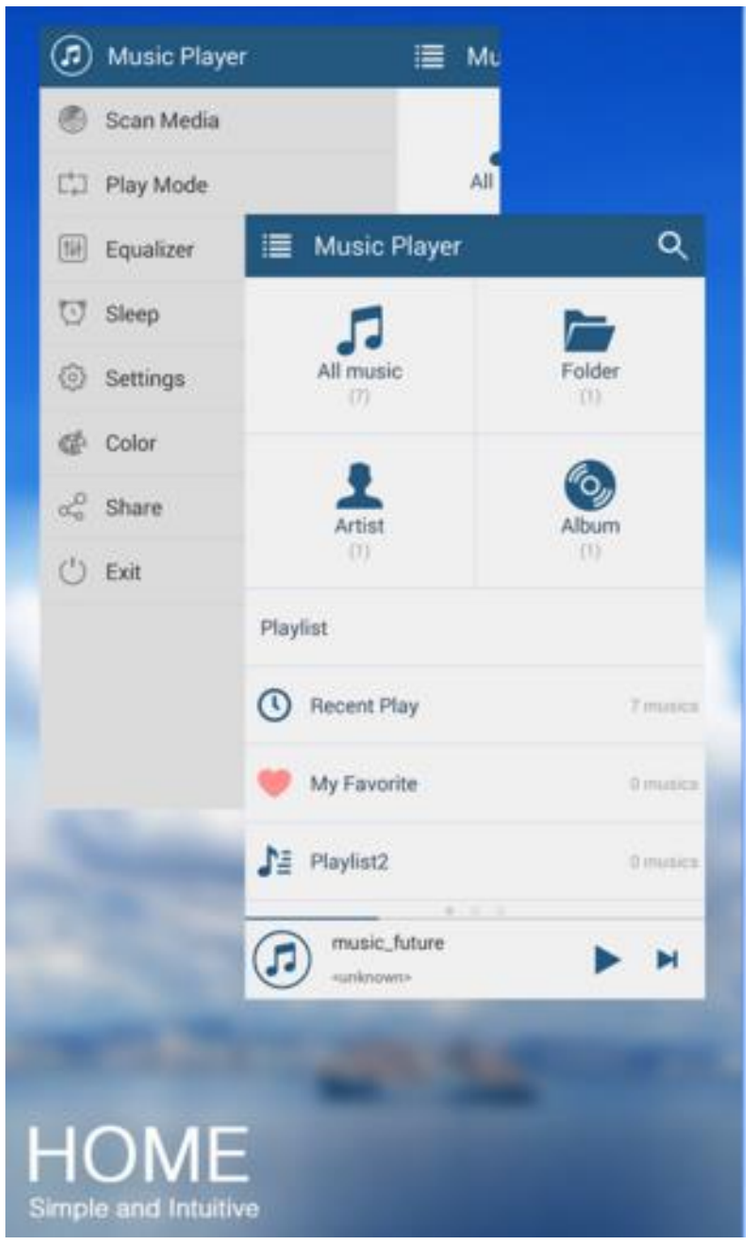
- Content Provider row updates are made with the Content Resolver update method:

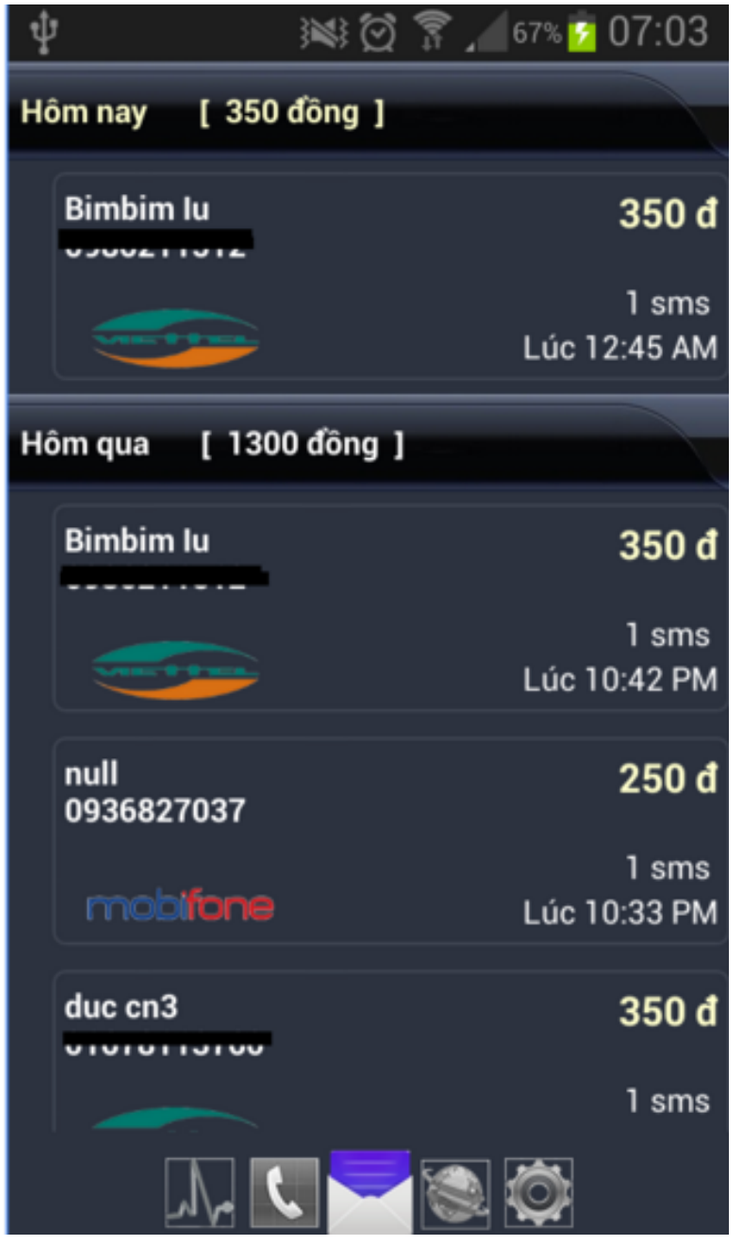
```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Create a replacement map, specifying which columns you want to
// update, and what values to assign to each of them.
newValues.put(COLUMN_NAME, newValue);
// Apply to the first 5 rows.
String where = "_id < 5";
getContentResolver().update(MyProvider.CONTENT_URI, newValues, where,null);
```

# Conclude



- Using Content Provider you can make music app, call time manager ..etc.



# Summary

- In this session, we learnt:
  - Content provider
  - Android's Built-in Providers
  - Architecture of Content Providers
  - Creating content provider
  - Exposing Access to the Data Source
  - Using content providers
  - Using the Cursor
  - Querying for Content
  - Adding, Updating, and Deleting Content