Android Data Storage

Saving content of your app

Storage Options

- **External Storage** - store public data on the shared external storage
- **Shared Preferences** - store private primitive data in key-value pairs
- **Internal Storage** - store private data on the device memory
- **SQLite Databases** - store structured data in a private database.
- **Network Connection** -store data on the web

External Storage

External Storage

- Useful to store downloaded data
- Can be also used to store computed data, but storing in DB is preffered
- Will not work on your emulator if no SD card memory is set
- Path to external cache can be gain by getExternalCacheDir() from any Activity or Context

Code Example

```
try {
        File externalCacheDir = getExternalCacheDir();
        File newFile = new File(externalCacheDir, “file.txt");

        BufferedWriter writer = new BufferedWriter(new FileWriter(newFile));
        writer.write("Hello World");
        writer.close();

        BufferedReader reader = new BufferedReader(new FileReader(newFile));
        String line = reader.readLine();
        reader.close();
        button.setText(line);
} catch (IOException e) {
        Log.e("Error", e.getLocalizedMessage());
}
```

AndroidManifest.xml

Do not forget to add this permissions to AndroidManifest!

<uses-permission
android:name="android.permission.**WRITE_EXTERNAL_STORAGE**"/>

<uses-permission
android:name="android.permission.**READ_EXTERNAL_STORAGE**"/>

SharedPreferences

SharedPreferences

- Allows you to save and retrieve persistent **key-value pairs**

- You can save any primitive data: booleans, floats, ints, longs, and strings.

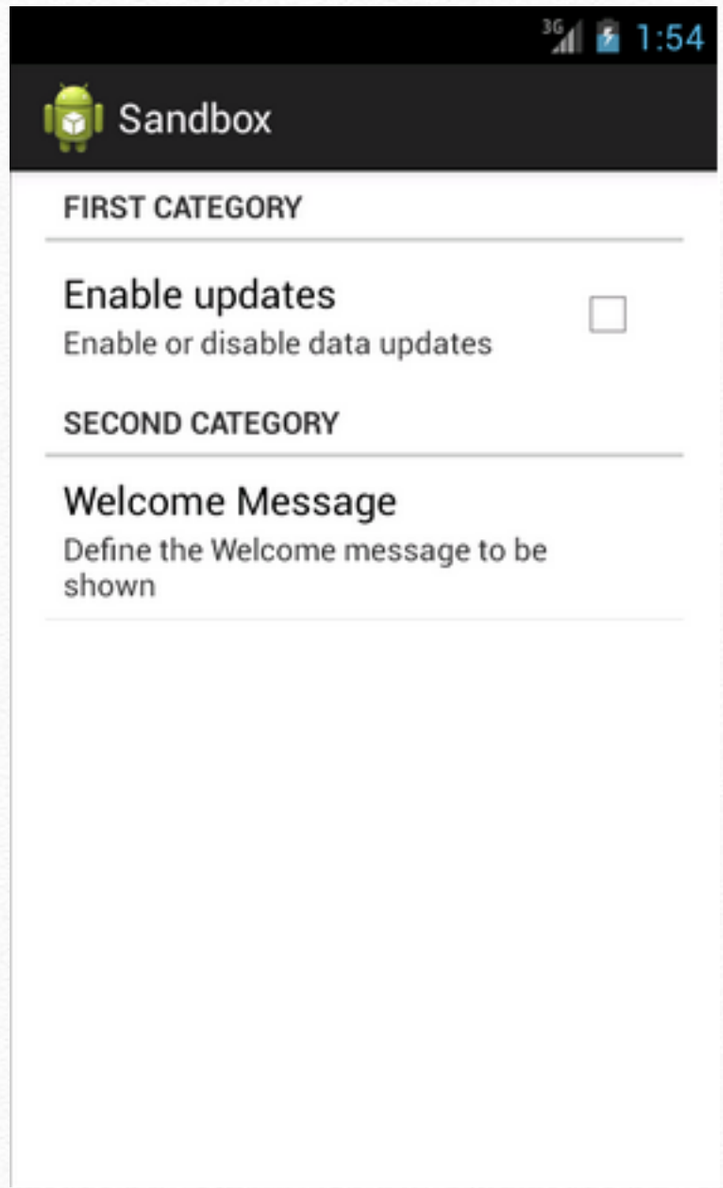
- Allow storing arrays and serialised objects

Example

```
SharedPreferences preferences = getSharedPreferences("MyPref", 0);
Map<String,?> all = preferences.getAll();
Log.d("Preferences", "" + all.size());

if(all.size() == 0 ){
    SharedPreferences.Editor edit = preferences.edit();
    edit.putString("hello", "world");
    edit.apply();
}

for(String s: all.keySet()){
    Log.d("Preference", s);
}
```

Preferences Activity

```
class MySettingActivity extends PreferenceActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.settings);
    }
}
```



http://viralpatel.net/blogs/android-preferences-activity-example/

Default Shared Pref

```
SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(this);
Map<String,?> all = preferences.getAll();
Log.d("Preference", "" + all.size());
for (String s : all.keySet()) {
    Log.d("Preference", s);
}
```

That store preference from PreferenceActivity

Internal Storage

Internal Storage

- **Private data** on the device memory.
- **Private to your application** and other applications cannot access them (nor can the user).
- When the user uninstalls your application, these files are removed.

Internal Storage

```
String FILENAME = “hello_file.txt";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
 fos.write(string.getBytes());
 fos.close();
```

MODE_PRIVATE will create the file (or replace a file of the same name) and make it private to your application.

Other modes available are:
MODE_APPEND, **MODE_WORLD_READABLE**,
and **MODE_WORLD_WRITEABLE**.

Internal Storage

```
final String FILENAME = "hello_file";
try{
        FileInputStream fileInputStream = openFileInput(FILENAME);
        StringBuilder builder = new StringBuilder();
        while(fileInputStream.available() > 0){
            builder.append((char)fileInputStream.read());
        }
        Log.e("Preference", builder.toString());
}
```

Database

Database

- Android has **SQLite** database out of the box.
- SQLite supports **standard relational database** features like **SQL syntax**
- Requires only **little memory at runtime**
- Any databases you create will be **accessible** by name to any class **in the application**, but not outside the application.
- SQLite **supports** the data types **TEXT**, **INTEGER** and **REAL**.

```
public class DatabaseHandler extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "contactsManager";
    private static final String TABLE_CONTACTS = "contacts";

    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "(
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_PH_NO + " TEXT" + ")";
        db.execSQL(CREATE_CONTACTS_TABLE);
    }
  ......
}
```

```
.....
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

        // Create tables again
        onCreate(db);
    }
.....
```

```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

```
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

```
// Getting single contact
public Contact getContact(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTACTS,
      new String[] {KEY_ID, KEY_NAME, KEY_PH_NO },
        KEY_ID + "=?",new String[] { String.valueOf(id) },
        null, null, null, null);

    if (cursor != null)
        cursor.moveToFirst();

    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    // return contact
    return contact;
}
```

Query params

Parameter	Comment
String dbName	The table name to compile the query against.
String[] columnNames	A list of which table columns to return. Passing "null" will return all columns.
String whereClause	Where-clause, i.e. filter for the selection of data, null will select all data.
String[] selectionArgs	You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array.
String[] groupBy	A filter declaring how to group rows, null will cause the rows to not be grouped.
String[] having	Filter for the groups, null means no filter.
String[] orderBy	Table columns which will be used to order the data, null means no ordering.

Query params

query(String table, String[] columns, String selection,
String[] selectionArgs, String groupBy, String having, String orderBy, Strin
limit)

db.query(TABLE_CONTACTS, new String[] {KEY_ID, KEY_NAME,
KEY_PH_NO }, KEY_ID + "=?",new String[] { String.valueOf(id) }, null, n
null, null)

"_id=19 and summary=?"

Cursor

Cursor **cursor** = **getReadableDatabase().**
rawQuery("select * from todo where _id = ?", new String[] { id });

- Represents the **result of a query** to one row of the query result
- **Buffer** the **query results** efficiently
- Has **getCount**(), **moveToFirst**(), **moveToNext**(), **TheisAfterLast**() methods
- Cursor provides typed **get***() methods, e.g. **getLong(columnIndex)**, **getString(columnIndex)** to access the column data for the current position of the result.
- getColumnIndexOrThrow(String) method allows you to get the column index for a column name of the table.
- A Cursor needs to be closed with the close().

```
public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<Contact>();
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }
    return contactList;
}
```

```
// Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

```
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

```
// Deleting single contact
public void deleteContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
    db.close();
}
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    DatabaseHandler db = new DatabaseHandler(this);

    // Inserting Contacts
    Log.d("Insert: ", "Inserting ..");
    db.addContact(new Contact("Ravi", "9100000000"));
    db.addContact(new Contact("Srinivas", "9199999999"));

    Log.d("Reading: ", "Reading all contacts..");
    List<Contact> contacts = db.getAllContacts();

    for (Contact cn : contacts) {
        String log = "Id: "+cn.getID()+
                        " ,Name: " + cn.getName() +
                        " ,Phone: " + cn.getPhoneNumber();
            Log.d(“Result”, log);
}
```

```
private void addCursor() {
    // Get all of the contacts from the database
    Cursor c = databaseHandler.getCursor();

    String[] from = new String[] { NotesDbAdapter.KEY_NAME };
    int[] to = new int[] { R.id.text1 };

    // Now create an array adapter and set it to display using our row
    SimpleCursorAdapter contacts =
        new SimpleCursorAdapter(this, R.layout.contacts_row, c, from, to);
    setListAdapter(contacts);
}
```

https://gist.github.com/LArchaon/5322063