Android Saving Data

Android applications save data in shared preferences. This data is only available to the application that saved it not to other applications running on the same device.

Here an EditText is used to capture data and hold it in a `SharedPreferences` object. This object is designed to hold application settings such as colours and scores but can be used to save small amounts of data for any purpose.

```
private EditText eInput;

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

eInput = (EditText)findViewById(R.id.txtInput);
// a file will be created named myFile the mode is 0 or
//private
SharedPreferences settings =
getSharedPreferences("myFile",MODE_PRIVATE);
        //settings will load a string from the key
//myFile if one exists
        eInput.setText(settings.getString("myFile ",""));
}
```

In the code above it is the `SharedPreferences` object that is used to persist data. This is placed in a key named `myFile`. The default mode restricts access to that string to this application only. 0 can be used as a shorthand instead of *`MODE_PRIVATE`*. To store more than 1 set of data the same `SharedPreferences` object can be used with different key values, take care over which keys are read and written to.

```
settings = getSharedPreferences("myScore",MODE_PRIVATE);

settings = getSharedPreferences("myImage",MODE_PRIVATE);
```

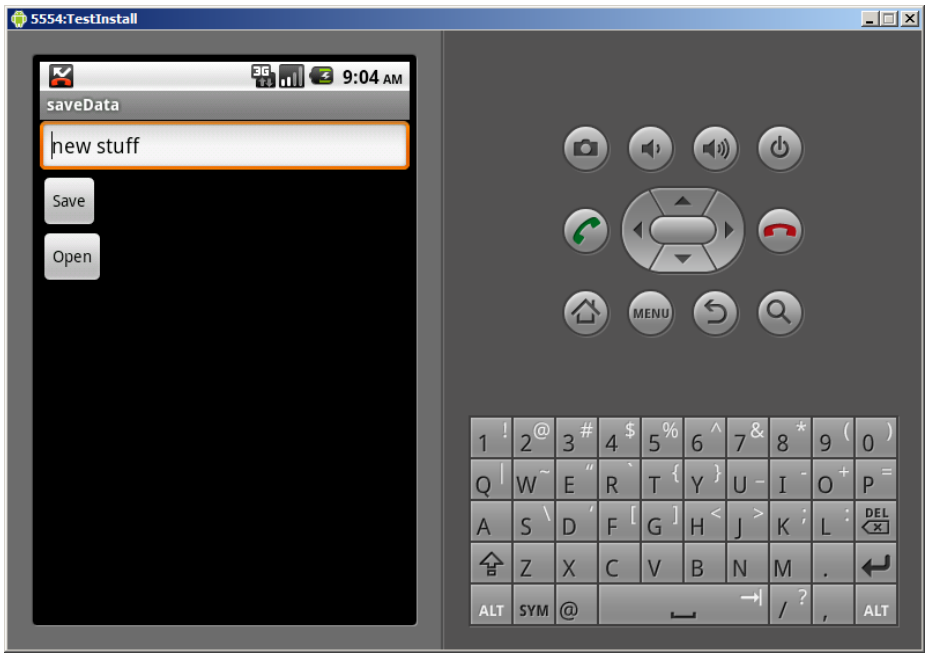
The `getString` method returns a string but there are similar methods within a `SharedPreferences` object to get int, float or Boolean values.

The method onStop is called when the application exits. This will be used to take the current text in the EditText and save it a `SharedPreferences` object.

```
 @Override
    //this method runs when the program exits
//do not put this method inside the onCreate method
 protected void onStop()
{
      super.onStop();
            SharedPreferences finalSettings =
      getSharedPreferences("myFile",MODE_PRIVATE);
      //create an Editor
      SharedPreferences.Editor editor = finalSettings.edit();
      //put the string from the EditText into the editor
      editor.putString("myFile", eInput.getText().toString());
      //commit the changes
      editor.commit();
}
```

The 2 methods use different `SharedPreferences` objects, settings and finalSettings but both use the same string as a key; “myFile”.

To make the program fire onStop it must be exited not paused. This function depends on the device. On this emulator the return button kills the application but the menu button does not.



In the example above 2 Buttons are used to save and return data while the program is running. It is best to also run the onStop() method to handle any possible mess ups when the application exits.

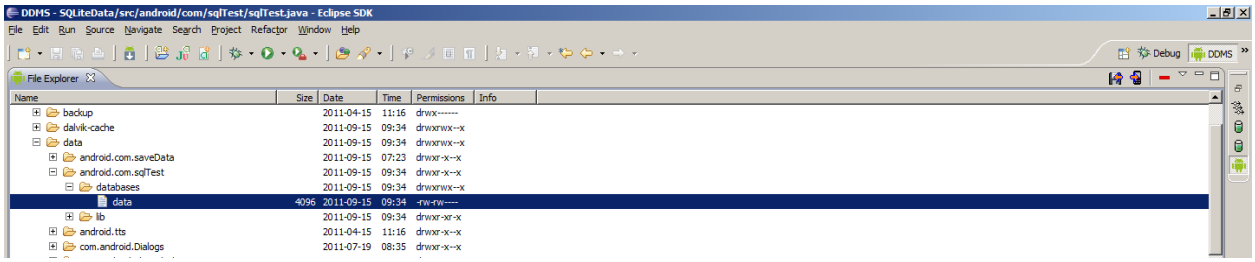
Using SQLite with Android

SQLite is a SQL database that is bundled into Android. This can be manipulated by passing SQL commands from the java code. This SQL will run onCreate it creates a new database, adds a table to that database and inserts 1 record into that table.

```
//open an existing database or create a new one
//private to this app
SQLiteDatabase myDb =
openOrCreateDatabase("data",MODE_PRIVATE,null);
//the int id can have up to 3 chars
myDb.execSQL("CREATE TABLE IF NOT EXISTS people(id
INT(3),lastName VARCHAR, firstName VARCHAR);");
myDb.execSQL("INSERT INTO people VALUES (1, 'Jones',
Bill');");
myDb.close();
```

The SQL commands are inserted as strings ending in a semi-colon hence the 2 semi colons on each line one inside and one outside the string. When passing VARCHAR data to SQL tables these values need to be in quotes (‘ ‘). Numeric data such as the id are not placed within quotations.

Not much appears to happen when this code is run but a file can be found within the file explorer of DDMS that corresponds to the new database.



It is unlikely that there will be only 1 record in a table. When returning data from a table it is assigned to a collection called a Cursor that can be iterated through.

Here the onClick() method is linked to a Button that will display the 2nd field of the 1st and only record of the database. The output is displayed on a TextView called lTest.

```
SQLiteDatabase myDb =
openOrCreateDatabase("data",MODE_PRIVATE,null);

String theQuery = "SELECT * FROM people;";

Cursor dataSet = myDb.rawQuery(theQuery, null);
dataSet.moveToFirst();
//column 1 - the 2nd column is a string
String dName = dataSet.getString(1);
//this will also work
```

```
 //String dName =
dataSet.getString(dataSet.getColumnIndex("lastName"));
myDb.close();
lTest.setText(dName);
```

The above code will work for a String. The 1st field of the people table is an int so the code to retreive it needs to be modified as follows.

```
int num = dataSet.getInt(0);
String dName =Integer.toString(num);
```

Working with SQLite

Any database reliant application can fail either due to the code or the underlying SQL commands. SQLite will run as a stand alone command line application on Linux, Mac and Windows. It makes sense to use this environment to test SQL commands and copy them into Eclipse if the results are satisfactory. SQLite as a stand along application is much quicker than running Eciplse and this will isolate SQL errors from other code errors.

The SQLite SQL syntax is outlined here. Creating a table, the TEXT datatype is the same as VARCHAR:

```
CREATE TABLE IF NOT EXISTS CDs (CDID INTEGER PRIMARY KEY,
ArtistID INTEGER NOT NULL,
Title TEXT NOT NULL,
Date TEXT);
```

The above is not a well structured table as it has no unique key so the same record can be entered several times. This does, however, prevent errors with the inserting data code below.

Inserting data:

```
INSERT INTO CDs VALUES (1 , 7 , 'Carpenters Greatest Hits',
'12-12-2011');
```

Updating an existing record:

```
UPDATE CDs SET ArtistID = 12 WHERE Title= 'Carpenters Greatest
Hits';
```

Delete a record:

```
DELETE FROM CDs WHERE Title LIKE 'Carpenters%';
```

Outputting several records from the SQL

To start some widget will be needed to show the data retreived. A simple solution is ot use a TextView but output each record on another line. This xml will set up a 5 line TextView that allows multi-line.

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Output"
    android:gravity="top"
    android:singleLine="false"
    android:lines="5"
    android:id="@+id/lTest"
  />
```

If a TextView exists called lTest and data has been placed in the Strings dName, sLast and sFirst the following code will place that data between commas on 1 line of the TextView. The next line will contain the words “next record”. The ‘\n’ character is used to denote a new line on the String

```
lTest.setText(dName + "," + sLast + "," + sFirst + '\n');
lTest.setText(lTest.getText()+ "next record");
```

We do not know how many records will be returned to the Cursor object so need to use a while loop to iterate through the collection until the last record is reached. Here the records are assigned to a String with a ‘,’ between each field and a new line after each record.

```
Cursor dataSet = myDb.rawQuery(theQuery, null);
dataSet.moveToFirst();

String finalData="";

while (dataSet.isAfterLast()==false)
{
      int num = dataSet.getInt(0);
      String dName =Integer.toString(num);
      String sLast = dataSet.getString(1);
      String sFirst = dataSet.getString(2);
      finalData = finalData + (dName + "," + sLast + "," +
sFirst + '\n');
      dataSet.moveToNext();
}
myDb.close();
```

This has happened because there is no unique key in the SQLite database.

This avoids duplicate data entry errors but would not be much use in the real world.

Tidying up the database

As the database structure and the records persist when the program is closed down the app is going to retain any messed up table definitions or records. The following code will delete (DROP) a table and all its records. If a table has been dropped it will need to be created again to enter any records. An alternative would be to alter a table. This will retain the records within the table if the fields match the new table structure. Changing a field from text to number for example may corrupt the table. Setting a unique key when several records share the same data for that field (such as the id in the app above) will fail.

This code will remove the table people from a database, the operation cannot be undone.

```
myDb.execSQL("DROP TABLE people");
```

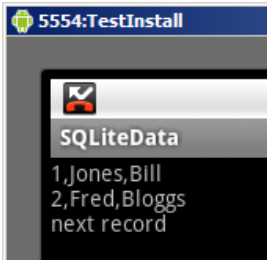
To create a better people table with a unique key so that each record is different the following CREATE code can be used.

```
myDb.execSQL("CREATE TABLE IF NOT EXISTS people(id INTEGER
PRIMARY KEY,lastName VARCHAR, firstName VARCHAR);");
```

The PRIMARY KEY flag is autoincrement by default so a value should not be assigned to it on the INSERT operation.

These INSERT lines add data to the lastName and firstName fields only, the id value is assigned by SQLite.

```
myDb.execSQL("INSERT INTO people (lastName , firstName) VALUES
( 'Jones', 'Bill');");
myDb.execSQL("INSERT INTO people (lastName , firstName) VALUES
( 'Fred', 'Bloggs');");
```



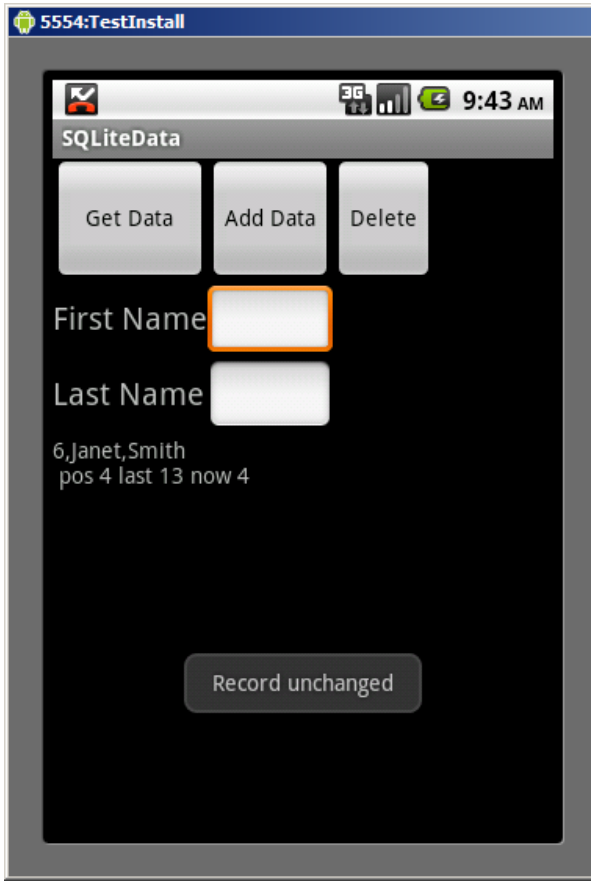
To add data at run time some controls will be required. Here an EditText is used for each. The adjacent TextView widgets have no code, their purpose is to guide the user as to what to put in the EditText boxes.

Here the 2 EditTexts have been assigned to the Android objects fName and lName. lTest is a TextView used for the output. The string sql is used to build up the SQL query string. This code checks that the query is properly formed. VARCHAR values are passed to the database within ‘ ‘ but numeric values are not.

```
String dataF = fName.getText().toString();
String dataL = lName.getText().toString();
String sql = "INSERT INTO people (lastName , firstName) VALUES
(";
sql = sql + "'" + dataF + "'" + ",'" + dataL + "');";
lTest.setText(sql);
```

The output of sql can be checked on screen and if it is acceptable can then be sent to the database.

```
SQLiteDatabase myDb =
openOrCreateDatabase("data",MODE_PRIVATE,null);
String sql = "INSERT INTO people (lastName , firstName) VALUES
(";
sql = sql + "'" + dataF + "'" + " , '" + dataL + "');" ;
myDb.execSQL(sql);
myDb.close();
```



Scrolling through records

The number of records that can be displayed in a single TextView is limited. This can be overcome by having the Get Data Button cycle through the records. Care must taken not to run past the last record in the database.

This navigation code depends on an int named pos to record the position within the database. The variable pos must be decalared at the top of the class with the Buttons and other widgets as it needs to retain its value between Button clicks. The record data is output as a String to the TextView lTest.

```
dataSet.moveToPosition(pos);
String finalData="";
int testPos = dataSet.getPosition();
int last = dataSet.getCount();
if(pos < last)
{
     int num = dataSet.getInt(0);
     String dName =Integer.toString(num);
     String sLast = dataSet.getString(1);
     String sFirst = dataSet.getString(2);
     finalData =  (dName + "," + sLast + "," + sFirst + '\n');
     lTest.setText(finalData + " pos " + pos + " last "  +
last + " now " + testPos);
     pos ++;
}
myDb.close();
```

Modifying individual records

The DROP command will drop the entire database but individual or sets of records can be changed by the UPDATE and SET command pair or by deleted by the DELETE command. The record to be removed or changed must exist. Record 32 cannot be removed if the records only run from 1 to 12.

In this code the position within the database returned by the scrolling routine is used to specify the record to be deleted.

```
int lastPos = pos-1; //incremented by in the navigate method
String sql = "DELETE FROM people WHERE id = " + posLast + ";";
lTest.setText(sql);  //test line
myDb.execSQL(sql);
myDb.close();
```

To allow a user to cancel the deletion or change of a record an AlertDialog can be used.

```
AlertDialog deleteAlert = new
AlertDialog.Builder(this).create();
deleteAlert.setTitle("Delete item");
int lastPos = pos-1;
deleteAlert.setMessage("Are you sure you want to delete item "
+ lastPos + "?");
deleteAlert.setButton("OK", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface
                    dialog, int which) {
                       //code to delete or update record
                     } });
deleteAlert.setButton2("No", new
DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog,
                  int which) {
                         //do not delete or update record
                } });
deleteAlert.show();
```