Part 4 - Using CursorAdapters

Android provides adapter classes specifically to display data from an SQLite database query:

SimpleCursorAdapter – Similar to an ArrayAdapter because it can be used without subclassing. Simply provide the required parameters (such as a cursor and layout information) in the constructor and then assign to a ListView.

CursorAdapter – A base class that you can inherit from when you need more control over the binding of data values to layout controls (for example, hiding/showing controls or changing their properties).

Cursor adapters provide a high-performance way to scroll through long lists of data that are stored in SQLite. The consuming code must define an SQL query in a Cursor object and then describe how to create and populate the views for each row.

Creating an SQLite Database

To demonstrate cursor adapters requires a simple SQLite database implementation. The code in SimpleCursorTableAdapter/VegetableDatabase.cs contains the code and SQL to create a table and populate it with some data. The complete VegetableDatabase class is shown here:

```
class VegetableDatabase  : SQLiteOpenHelper {
   public static readonly string create_table_sql =
       "CREATE TABLE [vegetables] ([_id] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE, [name] TEXT NOT NULL
UNIQUE)";
   public static readonly string DatabaseName = "vegetables.db";
   public static readonly int DatabaseVersion = 1;
   public VegetableDatabase(Context context) : base(context, DatabaseName, null, DatabaseVersion) { }
   public override void OnCreate(SQLiteDatabase db)
   {
       db.ExecSQL(create_table_sql);
       // seed with data
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Vegetables')");
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Fruits')");
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Flower Buds')");
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Legumes')");
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Bulbs')");
       db.ExecSQL("INSERT INTO vegetables (name) VALUES ('Tubers')");
   }
   public override void OnUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
   {   // not required until second version :)
       throw new NotImplementedException();
   }
}
```

The VegetableDatabase class will be instantiated in the OnCreate method of the HomeScreen activity. The SQLiteOpenHelper base class manages the setup of the database file and ensures that the SQL in its OnCreate method is only run once. This class is used in the following two examples for SimpleCursorAdapter and CursorAdapter.

The cursor query *must* have an integer column _id for the CursorAdapter to work. If the underlying table does not have an integer column named _id then use a column alias for another unique integer in the RawQuery that makes up the cursor. Refer to the for further information.

Creating the Cursor

The examples use a RawQuery to turn an SQL query into a Cursor object. The column list that is returned from the cursor defines the data columns that are available for display in the cursor adapter. The code that creates the database in the SimpleCursorTableAdapter/HomeScreen.cs OnCreate method is shown here:

```
vdb = new VegetableDatabase(this);
cursor = vdb.ReadableDatabase.RawQuery("SELECT * FROM vegetables", null); // cursor query
StartManagingCursor(cursor);
// use either SimpleCursorAdapter or CursorAdapter subclass here!
```

Any code that calls StartManagingCursor should also call StopManagingCursor. The examples use OnCreate to start, and OnDestroy to close the cursor. The OnDestroy method contains this code:

```
StopManagingCursor(cursor);
cursor.Close();
```

Once an application has a SQLite database available and has created a cursor object as shown, it can utilize either a SimpleCursorAdapter or a subclass of CusorAdapter to display rows in a ListView.

Using SimpleCursorAdapter

SimpleCursorAdapter is like the ArrayAdapter, but specialized for use with SQLite. It does not require subclassing – just set some simple parameters when creating the object and then assign it to a ListView's Adapter property.

The parameters for the SimpleCursorAdapter constructor are:

Context – A reference to the containing Activity.

Layout – The resource ID of the row view to use.

ICursor – A cursor containing the SQLite query for the data to display.

From string array – An array of strings corresponding to the names of columns in the cursor.

To integer array – An array of layout IDs that correspond to the controls in the row layout. The value of the column specified in the from array will be bound to the

The from and to arrays must have the same number of entries because they form a mapping from the data source to the layout controls in the view.

The SimpleCursorTableAdapter/HomeScreen.cs sample code wires up a SimpleCursorAdapter like this:

```
// which columns map to which layout controls
string[] fromColumns = new string[] {"name"};
int[] toControlIDs = new int[] {Android.Resource.Id.Text1};
// use a SimpleCursorAdapter
listView.Adapter = new SimpleCursorAdapter (this, Android.Resource.Layout.SimpleListItem1, cursor,
       fromColumns,
       toControlIDs);
```

SimpleCursorAdapter is a fast and simple way to display SQLite data in a ListView. The main limitation is that it can only bind column values to display controls, it does not allow you to change other aspects of the row layout (for example, showing/hiding controls or changing properties).

Subclassing CursorAdapter

A CursorAdapter subclass has the same performance benefits as the SimpleCursorAdapter for displaying data from SQLite, but it also gives you complete control over the creation and layout of each row view. The CursorAdapter implementation is very different from subclassing BaseAdapter because it does not override GetView, GetItemId, Count or this[] indexer.

Given a working SQLite database, you only need to override two methods to create a CursorAdapter subclass:

BindView – Given a view, update it to display the data in the provided cursor.

NewView – Called when the ListView requires a new view to display. The CursorAdapter will take care of recycling views (unlike the GetView method on regular Adapters).

The adapter subclasses in earlier examples have methods to return the number of rows and to retrieve the current item – the CursorAdapter does not require these methods because that information can be gleaned from the cursor itself. By splitting the creation and population of each view into these two methods, the CursorAdapter enforces view re-use. This is in contrast to a regular adapter where it's possible to ignore the convertView parameter of the BaseAdapter.GetView method.

Implementing the CursorAdapter

The code in CursorTableAdapter/HomeScreenCursorAdapter.cs contains a CursorAdapter subclass. It stores a context reference passed into the constructor so that it can access a LayoutInflater in the NewView method. The complete class looks like this:

```
public class HomeScreenCursorAdapter : CursorAdapter {
   Activity context;
   public HomeScreenCursorAdapter(Activity context, ICursor c)
       : base(context, c)
   {
       this.context = context;
   }
   public override void BindView(View view, Context context, ICursor cursor)
   {
       var textView = view.FindViewById<TextView>(Android.Resource.Id.Text1);
       textView.Text = cursor.GetString(1); // 'name' is column 1 in the cursor query
   }
   public override View NewView(Context context, ICursor cursor, ViewGroup parent)
   {
       return this.context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, parent, false);
   }
}
```

Assigning the CursorAdapter

In the Activity that will display the ListView, create the cursor and CursorAdapter then assign it to the list view.

The code in the CursorTableAdapter/HomeScreen.csOnCreate method is shown here:

```
vdb = new VegetableDatabase(this);
cursor = vdb.ReadableDatabase.RawQuery("SELECT * FROM vegetables", null); // cursor query
StartManagingCursor(cursor);
listView.Adapter = (IListAdapter)new HomeScreenCursorAdapter(this, cursor);
```

The OnDestroy method contains the StopManagingCursor method call described previously.

[<< Part 3 - Customizing a ListView's Appearance] [Part 5 - Using a ContentProvider >>]