


# Make Tasks Functional

Now that you have designed the interface for Tasks, it's time to make it a functional, useful application. You will construct a `ContentProvider` which will store tasks and provide an interface to the UI for accessing them. You will also construct a `Service` which handles maintenance of these tasks by deleting old, completed tasks.

Although you could include these components as part of the Tasks application you built in Assignment 1, to insure that everyone's UI is working properly, you will use a provided application, `TaskFrontend.apk`, which contains the `TaskList` and `TaskEdit` Activities. These Activities will expect your `ContentProvider` and `Service` to work as described below; if it does, then the UI will function and you will be able to create, modify, and delete tasks. The source for this will be released once everyone has submitted Assignment 1.

To install `TaskFrontend.apk`, start the emulator you wish to install the application on, and then from the command line, type `adb install <path/to/TaskFrontend.apk>`. `adb` is the debug interface for Android. It is a very useful tool which will let you perform many debugging tasks for your phone - you have already used `adb logcat`, which shows the device log. For more on `adb`, see <http://developer.android.com/guide/developing/tools/adb.html>. `adb` is located in the `tools/` directory wherever you installed the SDK. If the app does not install, try running `adb uninstall cornell.cs2046.tasks` first.

**New Android Project**  
Creates a new Android Project resource.



Project name: TaskBackend

Contents

- ☒ Create new project in workspace
- ☐ Create project from existing source
- ☒ Use default location

Location: C:/Users/Jeff/workspace/TaskBackend Browse...

Build Target

Target Name	Vendor	Platform	API ...
<input type="checkbox"/> Android 1.1	Android Open Source Project	1.1	2
<input checked="" type="checkbox"/> Android 1.5	Android Open Source Project	1.5	3
<input type="checkbox"/> Android 2.2	Android Open Source Project	2.2	8
<input type="checkbox"/> Google APIs	Google Inc.	2.2	8

Standard Android platform 1.5

Properties

Application name: TaskBackend

Package name: cornell.cs2046.taskbackend

☐ Create Activity:

Min SDK Version: 3

# Task Breakdown

As you did in the lab exercise, construct a new Android Java project in Eclipse to contain your `ContentProvider` and `Service`. Make sure your settings match exactly the ones in the above screenshot.

The keys are to name your project and application “TaskBackend”, use the Android 1.5 build target, the package “cornell.cs2046.taskbackend”, and the min SDK version 3. Make sure “Create Activity” is *not* checked. Please be sure you do not make a typo here; if your `ContentProvider` does not behave exactly as `Tasks.apk` expects, your `Tasks` will not appear in the provided UI.

## *Part 1: TaskProvider*

Part 1 of the assignment is to create the `ContentProvider` which stores and manages access to the tasks. Note that very little in this example is different from the simple Notepad provider that was explained in class; if you look at the sample code for this provider, posted on the course website, you should have very little trouble completing this part of the assignment. Unfortunately, while creating basic `ContentProviders` can be a bit tedious, this assignment will help you understand the basic infrastructure shared among all `ContentProviders` which use SQL.

First of all, copy the `Tasks.java` file into the `src/` folder of your workspace. This file contains constants like the column names that you should use for your table, and ensures that the interface exposed by your `ContentProvider` is exactly what is expected.

Create a new Java class in the `cornell.cs2046.taskbackend` package named `TaskProvider`. This class should extend `ContentProvider`, and must therefore implement the `onCreate`, `query`, `insert`, `update`, `delete`, and `getType` methods.

As a first step, you should create a `DatabaseHelper` inner class which provides access to the SQLite database which will store the tasks. See the example `DatabaseHelper` for the general approach you should take. Your `Tasks` table should contain these columns:

Column Name	Datatype
Tasks._ID	INTEGER PRIMARY KEY
Tasks.TITLE	TEXT
Tasks.DETAILS	TEXT
Tasks.COMPLETED	INTEGER
Tasks.DATE_MODIFIED	INTEGER

The `execSQL` command that you should call in `DatabaseHelper.onCreate` is the only raw SQL that you should have to type; as long as you model your command exactly after the class example, you should be fine. Note that there is no binary type in SQLite, so

Tasks.COMPLETED is an Integer. As described in the comments in the Task class, you should store incomplete tasks as 0's in this column and complete tasks as 1's.

Next, implement the simple functions in TaskProvider. `onCreate()` should create a new instance of DatabaseHelper to be used by the other methods of the ContentProvider. `getType()` can just return null; the Tasks UI does not bother to check the MIME type of the ContentProvider.

`query()` and `delete()` will work essentially identically to those methods in the class example. `update()` and `insert()` are similar, with one additional feature - in addition to using the ContentValues object that was passed in to modify the table, you should also set the DATE\_MODIFIED column to `System.currentTimeMillis()`. By doing this, your provider will automatically keep track of the last time a Task was modified, without requiring applications which use the provider to update this field themselves.

However, you should not modify the values object that was passed in as an argument directly. This is generally a bad practice - unless a function signature is documented otherwise, the input parameters should not be modified. For example, suppose the user was going to reuse this ContentValues object for something else. In this case, the object would contain an additional row, containing the current date, and this may not be what the user wants.

Thus, make sure you do not modify the initial ContentValues object. Instead, use the *copy constructor* that is part of the ContentValues class. A copy constructor for a class takes one argument; an instance of the class itself. It returns a new instance which is a copy of the original instance; hence, the name. This way, you can copy the input object, add the DATE\_MODIFIED value, and use this new object for any database calls, while leaving the initial instance unchanged.

Finally, add your provider to the AndroidManifest file, with `android:authorities="cornell.cs2046.taskbackend"`, which registers your provider with the Android system under that content:// URI.

## *Part 2: TaskService*

TaskService is a background service responsible for deleting completed tasks which are older than a week. ContentProviders and SQLite make this specific task relatively simple. Note that after completing this assignment, the only way that this service will be started is by pressing the Menu key and selecting "Delete Old". In Assignment 3, you will use the AlarmManager to run this service daily.

Create another new class named TaskService which extends Service. Eclipse will force you to provide an implementation of `onBind()` - our service will just perform work on demand, and so this method can and should just return null.

The work of the service should be done in the `onStart(Intent intent, int startId)` method. When the service receives a command, i.e. an Intent, with the action Tasks.ACTION\_DELETE\_OLD, it should spawn a new Thread and delete completed tasks

which are older than a month in this thread. Deleting is done with the `ContentResolver.delete()` method as described in class. You'll need to specify a WHERE clause describing the specific rows which should be deleted. From lecture, you should know how to restrict the search to completed tasks. For the week-old restriction, use the following condition:

```
datetime(" + Tasks.DATE_MODIFIED + "/1000, 'unixepoch') < datetime('now, '-7 days')";
```

`datetime` is an SQLite command; with the `unixepoch` parameter, it will return the current time in seconds - this is according to the same frame of reference as the time returned in `System.currentTimeMillis()`. We restrict the delete to rows in which the last modification was done earlier than 7 days ago, meaning the task is at least one week old.

Make sure your WHERE clause specifies *both* that the task is completed and that the task is at least a week old. If you want to test this method to make sure it works, replace -7 days with -5 seconds, modify a task, wait at least 5 seconds, and then make sure the task is deleted when you select "Delete Old", but only if the task is marked complete.

Don't forget to call `stopSelf` at the end of `onStart` to insure that the service does not stay activated while idle. Recall the method described in class and on the newsgroup of using a Queue of startIDs to insure that if multiple threads call `onStart` simultaneously, then the Service won't stop until all commands have fully executed.

Of course, you must also add your service to the AndroidManifest file. However, since `TaskList` is part of a different application, it does not have access to `TaskService.class`, and so it uses an implicit Intent to launch the service. Thus, make sure your service has an Intent filter specified which matches the `Tasks.ACTION_DELETE_OLD` action.

### *Part 3: Querying Contacts*

Parts 1 and 2 complete `Tasks` and make it a fully-functional application. However, since the revised implementations of `TaskList` and `TaskEdit` are provided for you, you have not yet built an application which queries a `ContentProvider`. For Part 3, we will make a separate, small Activity which simply displays the Call Log on the phone in a `ListView`.

We need a `ListView` layout, so modify `res/layout/main.xml`, which was automatically generated, and replace the `TextView` with a `ListView`. Don't worry about an empty View for this part.

Create a new Activity, `CallLogActivity`, as part of the `TaskBackend` project. In the `onCreate` method of this Activity, we need to:

- Query the `CallLog` provider. The Content URI is `CallLog.Calls.CONTENT_URI`. Your query should return three columns - `CallLog.Calls._ID`, `CallLog.Calls.NUMBER`, and `CallLog.Calls.DURATION`. Recall that you can use the projection argument of a query to restrict which columns the query returns.

One method that was not introduced in class is `managedQuery`. This method takes the same arguments as `query()`; however, when you use it, the Activity handles refreshing the data, closing the cursor when the Activity pauses, and restoring it when the Activity resumes. You may use this method to handle the Cursor's lifecycle.

Typically, these queries should be done in a background thread; however, for this part of the assignment, you may perform the queries on the main UI thread.

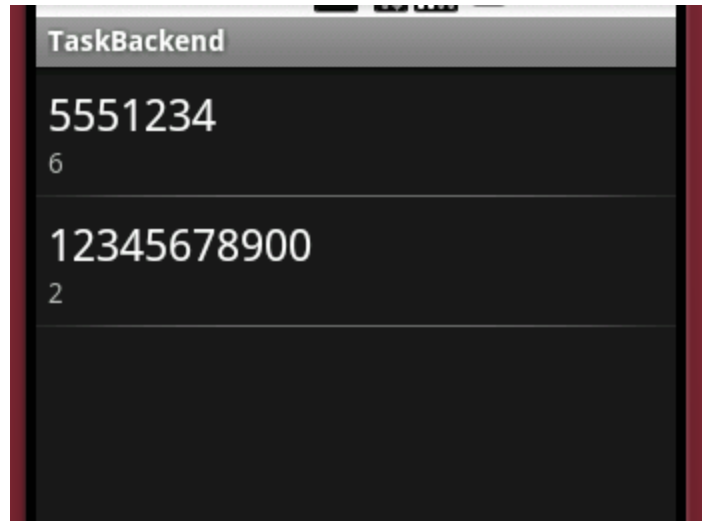
- Use a `SimpleCursorAdapter` to adapt the `CallLog` data to the `ListView`. For our purposes, this adapter will be much simpler than the one from Assignment 1. Use the following constructor:

```
String[] from = {CallLog.Calls.NUMBER, CallLog.Calls.DURATION};
int[] to = {android.R.id.text1, android.R.id.text2};
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_2, cursor, from, to);
```

Once again, we are using built-in layouts; the `simple_list_item_2` layout contains two `TextView`s, one primary and one secondary. The adapter binds data from the cursor from the columns specified in the `from[]` array to the views specified in the `to[]` array. If we had more complex data to bind, we could make a custom `ViewBinder` with the `setViewBinder()` method, which does similar work to the `getView` method you implemented in Assignment 1, but is a bit simpler to use. For an example of this, see the `TaskList` code which is provided to you.

- Set the `ListView`'s adapter to this `SimpleCursorAdapter` to display the call log.
- Add the Activity to the `AndroidManifest` file. In order to get the Activity to run, I needed to add an Intent filter which adds the Activity to the launcher. In addition, your manifest must specify the permission `android.permission.READ_CONTACTS` in order to be able to read from the Content Provider.

You can make fake calls in the emulator by pressing the green phone button below the screen, selecting the dialer tab, dialing a number, and pressing the phone button again. Hang up by pressing the red phone button. Then, run your Activity to display the call log. You should see something similar to this:



The first line is the number you dialed; the second is the duration of the call, in seconds.

## Some Tips

When testing your application, make sure you first install TaskBackend before running Tasks. You can just hit the Run button for the TaskBackend project and run it as an Android Application; this will install the .apk, but not launch any Activities.

To submit, zip up the entire TaskBackend project and submit the .zip file on CMS.

## Karma Problems

See the course website for an explanation of what these problems are. If you complete any of them, include a readme.txt file in your submission explaining what you have completed. If there's something else you'd like to try to do for karma credit, you can e-mail the instructor for approval and guidance.

In roughly increasing difficulty order:

- Add an additional query to the ContentProvider, `content://cornell.cs2046.taskbackend/tasks/last_updated`, which returns the task which was most recently updated.
- Extend your ContentProvider to support a Live Folder view of the tasks, so users can view the task list from the Home screen. If the user clicks a task, this should launch the TaskEdit activity for that task. You can add a dummy Activity which processes the Live Folder intent and then terminates, as described in lecture.
- Create an App Widget which displays the most recently modified task. You can either use the additional query from the first karma problem, or just use the regular interface.