CS 2046 - Assignment 3	**Due**: *Friday*, 11/19, 11:59pm on CMS

Connect Tasks to the Internet

After completing Assignment 2, your Tasks application is a fully-functional to-do list application, but it is standalone - there is no way of accessing the tasks except on the phone the application is installed on. In the real world, we expect that users will want to access their tasks not only from one phone, but from other phones they use, and from any computer connected to the internet. In this assignment, you will connect Tasks to an online service which backs up the task list.

The web service has been written for you. It is running on Google App Engine, for which more information is available at
http://code.google.com/appengine/docs/whatisgoogleappengine.html. The service is sparse, and supports fetching the last saved backup or posting a list of tasks to replace those which are currently saved. The service uses JSON for all communications. It does not require authentication, and it associates tasks with a given NetID which is passed as a parameter in all queries. The lack of authentication is for simplicity; please be respectful of other students by not using NetIDs other than your own, or otherwise violating the protocol for communicating with the API.

Once again, you will be provided with some components of Tasks so you can focus your efforts on backing up Tasks. A new version of TaskFrontend.apk is included with this assignment, and it includes TaskProvider and TaskService from Assignment 2. The UI has been updated to include menu buttons for manually backing up tasks and restoring the latest backup. There is also a new version of Tasks.java containing some additional constants relating to this assignment.

Task Breakdown

Similar to Part 2 of Assignment 2, you will once again be creating a Service to perform a background task. This time, the Service will be responsible for two commands - a backup command, which stores all of the tasks on the Backup server, and a restore command, which restores the tasks from the latest backup. You will also use AlarmManager to run the backup command daily, and a BroadcastReceiver to insure that even if the phone (or emulator) is restarted, that the backup command will still be executed.

First, you should examine the web service and understand how it works. Here is some example output from http://cs2046tasks.appspot.com/tasksync?netid=jpd236, reformatted for clarity:

```
{
  "tasks":[
    {
      "title":"Saved task",
      "lastModified":1289368052844,
      "details":"",
      "completed":false
    },
    {
      "title":"Saved Task 2",
      "lastModified":1289368963199,
      "details":"",
      "completed":true
    }
  ]
}
```

As you can see, these JSON objects contain essentially all of the information from the task list, apart from the ID column used internally by Android. When you restore from a backup, you will fetch the JSON object containing all of the tasks for your NetID and insert these into TaskProvider. Conversely, to save a backup, you will create the same JSON object for the tasks on the emulator and post this to the server.

To begin, exactly as you did in Assignment 2, create a new Android Java Project in Eclipse. Name this project and application TaskBackup, with package `cornell.cs2046.taskbackup`, and otherwise follow the Assignment 2 instructions.

Part 1: Backing up and Restoring Tasks

Create a new Service named TaskBackupService. This Service should be configured to respond to two distinct actions:

- `Tasks.ACTION_BACKUP_TASKS`: The service will back up all current tasks to the Internet. The following steps should be taken:
  - The service should query the Tasks provider (at `Tasks.CONTENT_URI` as usual) to obtain the current list of tasks.
  - The list of tasks should be converted to a `JSONObject`, as above. The name of the JSONArray containing each task is `Tasks.JSON_TASK_ARRAY`. The fields are `Tasks.JSON_TITLE`, `Tasks.JSON_DETAILS`, `Tasks.JSON_COMPLETED`, and `Tasks.JSON_DATE_MODIFIED`. See Lecture 7's slide about `Cursor`s for how to loop over a `Cursor`, as you will need to create a `JSONObject` for each task. See Lecture 10 and the org.json Android Javadoc for how to populate a JSON object - you can use `new JSONObject()` and `new JSONArray()` to create objects and arrays, and use the put method to add elements to each. When you are finished with your `Cursor`, be sure to call `close()` on it to free the database connection.

  - Perform an HTTP POST request to upload your tasks to the web service. You should use `HttpClient` as per Lecture 10 to complete this task. In your POST, you should connect to `Tasks.BACKUP_API_URL`, and pass two parameters:
    * `Tasks.REQUEST_NETID` - your NetID
    * `Tasks.REQUEST_TASKS` - the above JSON object, converted to a String using toString()

- `Tasks.ACTION_RESTORE_TASKS`: The service will restore the tasks which were saved in the most recent backup.
  - Start by deleting all of the Tasks in the database.
  - Perform an HTTP GET request using `HttpClient`. You should query this URL:

    ```
    Tasks.BACKUP_API_URL + "?" + Tasks.REQUEST_NETID + "=<your NetID>"
    ```

    To obtain the server's response, you can use the method in class to read the `InputStream` returned by `HttpClient` into a String. Alternatively, since the responses are small, you can use `EntityUtils.toString()` on `client.execute(getRequest).getEntity()`, which will read the entire response into a String.
  - Use `JSONTokener` to convert this string into a `JSONObject`, as per the example in Lecture 10.
  - Iterate over the `JSONArray` located inside of the `JSONObject`, inserting each Task into TaskProvider.

As in Assignment 2, your service should perform all work in a background thread and call `stopSelf` in such a manner that the Service will terminate once all work is completed. You will also need to add an entry to your AndroidManifest file for the service, such that it will respond to requests for both of the above actions. Don't forget to add the permission required to access the Internet.

Once the Service is completed, you should be able to successfully backup and restore Tasks using the provided TaskFrontend.

Part 2: Periodic Task Backup

Next, we want to insure that Tasks are backed up regularly. We do this using the AlarmManager class, which was explained at the end of Lecture 8. Example code for using AlarmManager is included in these lecture slides.

Create a new class named `TaskBackupReceiver` which extends `BroadcastReceiver`. Your overridden `onReceive` method needs to handle two different actions:

- Tasks.BROADCAST_PERIODIC_BACKUP. Upon receiving an Intent with this action, the receiver should simply start the backup service to perform a backup command.
- Intent.ACTION_BOOT_COMPLETED. This Intent is sent after the phone has turned on, and thus allows us to restore the alarm after a restart. If this action is received, reset the alarm for whatever time was last set in the `SharedPreferences`. If no time has been saved, do not set an alarm. Note that the AlarmManager will automatically trigger the alarm if the time is in the past, so you do not have to worry about this case.

The next step is to have the Service set the alarm for the next time it should run. Any time it performs backup, it should also set an alarm to start itself again in 24 hours. By using this approach, the tasks will be backed up every day, and if the user performs a manual backup, the next backup will occur 24 hours after that backup, instead of after the last periodic backup. The Intent which you pass to AlarmManager should perform a broadcast with TaskBackupReceiver as its component and the `Tasks.BROADCAST_PERIODIC_BACKUP` action. In addition, you should save the time for which you are setting the backup alarm in a `SharedPreferences` object.

Finally, to actually receive these events, be sure to add an intent filter for your receiver with both the "`cornell.cs2046.taskbackup.BROADCAST_PERIODIC_BACKUP`" and the "`android.intent.action.BOOT_COMPLETED`" action. You also need to declare the `android.permission.RECEIVE_BOOT_COMPLETED` permission - apps which run on startup have the potential for slowing down the boot time of the phone, so Android requires that apps which receive this event declare this permission at install time.

Be sure to test that your alarm manager and receiver are working by temporarily reducing the backup period, and by restarting the emulator and insuring that the correct alarm is reestablished.

Congratulations! Once you have completed this assignment, you will have designed a fully-fledged, internet-connected Android application, and you will have completed your requirements for CS 2046. I hope you will continue to create mobile applications for the platform of your choice in the future - if you create something and put it on the market, please do be sure to send me a link!

Some Tips

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

To submit, zip up the entire TaskBackup 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.

- Notifications. Currently, there's no real notification that backup or restore is occurring or has succeeded, except for a small Toast displayed by TaskList when the action is initiated. Use NotificationManager to display notifications when backup and restore are taking place, and change it once the action is complete. Clicking the notification should open the TaskList to view the list of Tasks.

  You might consider only showing the notification if the backup or restore causes some change, either to the client or the server, to avoid spamming the user each day.

- Implement restore transactions. When restoring from a backup, the assignment has you first delete all tasks, and then download the tasks from the server. This approach is extremely risky - if the phone is offline, then the resulting task list will be empty. Prevent this from happening by ensuring that one of two events always happens:

  - Restore fails, and the previous list of tasks is retained.
  - Restore succeeds, and the backed up list is restored.

- Backoff and retry. Implement a scheme where if the backup or restore fails, it will be retried in a reasonable fashion. One approach which was described in class is to start by retrying in a minute, and then doubling the amount of time between subsequent retries until one succeeds. You should probably cap the maximum retry at a reasonable period (perhaps a few hours), and stop retrying once the next period of regular updates is reached.