LOCATION BASED ALERTS ON ANDROID MOBILE SYSTEMS

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Sukanya Nagarajan

FALL
2012
LOCATION BASED ALERTS ON ANDROID MOBILE SYSTEMS

A Project

by

Sukanya Nagarajan

Approved by:

__________________________________, Committee Chair
Du Zhang, Ph.D.

__________________________________, Second Reader
Chung-E Wang, Ph.D.

___________________________
Date
Student: Sukanya Nagarajan

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Nikrouz Faroughi, Ph.D.

__________________________
Date

Department of Computer Science
Abstract

LOCATION BASED ALERTS ON ANDROID MOBILE SYSTEMS

by

Sukanya Nagarajan

The aim of this project is to create an application that can provide location based alerts on Android mobile phones.

While a time based alarm application alerts a user at a specified time, a location based alert application alerts a user when he/she is in the proximity of a specified location. The application lets the user specify a central location of a region and a radius around it and alerts the user when he/she enters the circular region. Both GPS and network provider can be used for providing location based alerts. Users can use GPS for getting more accurate location information and use network providers to preserve battery life and to get location information indoors. The following are the main features of the application. (1) Alert List. The application stores location alerts in file system and displays them every time the application is opened. User can add new alerts, edit/delete existing alerts and set multiple alerts for different locations. The application is not required to run in the foreground for receiving the alerts. (2) Weekly Reminders. User can choose to receive alerts on specific days of the week. (3) Favorite-places List. User can maintain a list of favorite places and use the list to enable location alerts for those places when needed. (4) Interaction with contacts-list. The application interacts with
native 'contacts' application and lets the user choose address of any of the contacts for setting a location alert. (5) Enable/Disable alerts. The application provides options to turn on/off alerts and to set radius.

Though there are existing mobile applications that provide location-based services, this application has some unique set of features that differentiates it from other applications. For instance, the application maintains a list of favorite places the user would like to visit and helps user set location alerts for the places in the list. The application further enhances user experience by letting user set location alerts for any of the contacts in native contact-list application. The application also helps in improving performance by implementing an efficient algorithm for setting weekly reminders that saves battery life.

The development environment consists of Windows 7 OS, Eclipse Java IDE with Android plug-ins, Sun Java SE Development Kit, Android 2.3.6 operating system (Gingerbread), Android framework API level 10 and Android based handset (Motorola Atrix 2).

_______________________, Committee Chair
Du Zhang, Ph.D.

_______________________
Date
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter</strong></td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. RELATED WORK</td>
<td>3</td>
</tr>
<tr>
<td>3. TECHNOLOGY</td>
<td>4</td>
</tr>
<tr>
<td>3.1 Android</td>
<td>4</td>
</tr>
<tr>
<td>3.1.1 Android Applications</td>
<td>5</td>
</tr>
<tr>
<td>3.1.2 Building Blocks of an Android Application</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Development Environment</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Location Technologies</td>
<td>9</td>
</tr>
<tr>
<td>4. METHODOLOGY</td>
<td>12</td>
</tr>
<tr>
<td>4.1 Features of the Application</td>
<td>12</td>
</tr>
<tr>
<td>4.2 Methodology</td>
<td>13</td>
</tr>
<tr>
<td>5. IMPLEMENTATION</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Geocoding</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Proximity Alerts</td>
<td>19</td>
</tr>
<tr>
<td>5.3 Persistence</td>
<td>22</td>
</tr>
<tr>
<td>5.4 Broadcast Event</td>
<td>23</td>
</tr>
<tr>
<td>5.5 Notifications</td>
<td>23</td>
</tr>
</tbody>
</table>
5.6 Address Input Options ................................................................. 24
5.7 Weekly Reminders Feature ............................................................ 26
5.8 GUI Screens .................................................................................. 27

6. FINDINGS, INTERPRETATIONS AND FUTURE WORK ......................... 37
Appendix Source Code ........................................................................ 40
Bibliography ......................................................................................... 82
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Process Priority Tree</td>
<td>5</td>
</tr>
<tr>
<td>2.</td>
<td>Steps Involved in Building and Running an Android Application</td>
<td>9</td>
</tr>
<tr>
<td>3.</td>
<td>Methodology for Setting Proximity Alerts</td>
<td>14</td>
</tr>
<tr>
<td>4.</td>
<td>A UML Use Case Diagram</td>
<td>21</td>
</tr>
<tr>
<td>5.</td>
<td>Alert-List Screen</td>
<td>28</td>
</tr>
<tr>
<td>6.</td>
<td>Edit-Alert Screen</td>
<td>29</td>
</tr>
<tr>
<td>7.</td>
<td>Location Input Options</td>
<td>30</td>
</tr>
<tr>
<td>8.</td>
<td>Edit-Address Dialog</td>
<td>31</td>
</tr>
<tr>
<td>9.</td>
<td>Alert-Radius Dialog</td>
<td>32</td>
</tr>
<tr>
<td>10.</td>
<td>Days-of-Week Dialog</td>
<td>33</td>
</tr>
<tr>
<td>11.</td>
<td>Favorite-Places-List Screen</td>
<td>34</td>
</tr>
<tr>
<td>12.</td>
<td>Add-Favorite-Place Screen</td>
<td>35</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The aim of this project is to create an application to provide location based alerts on Android mobile phones.

While a time based alarm application alerts a user at a specified time, a location based alert application alerts a user when he/she is in the proximity of a specified location. Location alert applications are quite useful when time of an event is unknown. The application uses Android mainly because it is open source and provides a complete software stack and tool set that make mobile application development easier.

The application lets the user specify address of central location of a region and a radius around it and alerts the user when he/she enters the circular region. It uses both Global Positioning System (GPS) and Network Location Provider to acquire user location information. GPS, a satellite based navigation system provides location information to Android devices that come with built in GPS receivers. Network Location Provider on the other hand uses cell tower and Wi-Fi signals to provide location information.

Location based alerts have several day-to-day applications. For instance, when a person gets into a train or bus, he/she can provide address of the destination station to the application and engage himself/herself in any activity (read books, take nap etc.). When the bus or train reaches within the specified radius of the destination, the application beeps and alerts the user. This way the application helps the user realize that he/she is somewhere near to the destination and the user does not miss his destination.
The project is explained in more detail in the following chapters. Chapter 2 discusses related work, Chapter 3 covers the technology aspects, Chapter 4 talks about the methodology used, Chapter 5 covers the implementation details and Chapter 6 discusses findings, interpretations and future work.
Chapter 2

RELATED WORK

There are several mobile applications that provide location based services on leading mobile devices such as Android, iPhone and Windows. These applications differ in GUI, features and functionalities and users generally select an application from this pool based on their requirements. The application being developed has a set of features that differentiate it from many of the existing location alert applications including Geo-Reminders (iPhone) [8], Proximity Alert (Android) [7] and Trapster (Windows) [6]. Some of the main features of these applications include displaying a map to select reminder location, listing previously created reminders in a map and sending sms to user on reaching a reminder location.

Features like weekly reminders, interaction with native contacts application for setting alerts and maintenance of favorite places list distinguishes this application from the above mentioned applications. Weekly reminder feature lets user set location alerts that are repeated every week on user specified days when he/she approaches the specified destination. By interacting with native contacts application, the application lets the user set alerts for addresses in contact list. It has an option to display contact-list and when user selects a contact in the list, it sets alert for the corresponding contact address. The application also lets user create a list of his favorite places for which he/she can set alerts when needed. User can add new places to the list and also edit and delete existing places in the list. The application further helps in improving performance by implementing an efficient algorithm for setting weekly reminders that saves battery life.
Chapter 3

TECHNOLOGY

3.1 Android

Android is an open platform for mobile devices developed by Google and Open Handset Alliance [5]. It is a Linux based operating system with open source libraries for application development. One of the unique features of Android is that it does not differentiate native applications from third party applications. While most mobile phone systems prioritize native applications over third party applications, Android executes both types of applications on the same run time and provides common set of APIs for both native and third party applications. Similar to native applications, third party applications in Android have access to hardware components through Android libraries.

Android software stack includes Linux Kernel, Libraries, Android run time, Application framework and native and third party applications. Linux kernel acts as an interface between hardware and software components and device drivers are part of kernel. The kernel also takes care of low level management including process management, memory management and power management. The next layer on the software stack consists of wide range of libraries including SSL, SQLite and OpenGL. On top of libraries is Android run time that includes Android core libraries and Dalvik Virtual Machine. The libraries in Android are Java based and includes most of the Java APIs as well. Application framework includes APIs and classes that are used for developing native and third party applications [1].
3.1.1 Android Applications

Mobile applications are software programs installed on mobile devices that perform a variety of tasks to help users with their day-to-day activities. Categories of mobile applications include internet applications, multimedia applications, gaming applications, navigation applications and security applications. The wide range of libraries and tools provided by Android enable mobile application developers to build efficient and rich applications with a number of features.

![Process Priority Tree](image)

**Figure 1. Process Priority Tree**

Each Android application runs in its own process and Android runtime takes care of managing the resources for the processes. Android prioritizes all the processes and whenever there is a memory constraint, it eliminates processes with lowest importance. As it can be seen from Figure 1 [1], Android categorizes the processes in order of importance as Foreground processes, Visible processes, Service processes, Background processes and Empty processes with foreground processes having the highest priority and
empty processes having the lowest priority [5]. Foreground processes refer to processes that need user interaction, broadcast receivers (discussed in section 3.1.2) executing `onReceive` method and services that have priority equivalent to foreground processes. Visible processes refer to visible activities that are hidden partially and are not running in the foreground. Service processes refer to processes that run without a user interface. Background processes refer to activities that run in the background hidden by another activity and empty processes refer to closed applications that are retained by the system in cache memory.

3.1.2 Building Blocks of an Android Application

Manifest File

One of the most important components of an Android application is a manifest file. A manifest file defines everything about an Android application and the system uses this manifest file for executing an application. A manifest file includes the libraries an application need, permissions required by the application to interact with other applications, API level used and software/hardware features required by the application.

Activities

Activity in general represents a user interface screen presented by an application. Starting an activity opens a new screen and returning from an activity takes the user back to the old screen. Android maintains an Activity Stack that contains all the currently running activities. The stack uses Last-In-First-Out principle [1]. Whenever a new activity is opened, the current activity is moved to the top of the stack and when the new activity is closed, the last activity from the top of the stack is displayed to the user. All
activities of an application are registered in the manifest file. Android system displays an activity only if it is registered in the manifest file.

**Intents**

Intents provide a mechanism to make different applications in an Android device interact with one another. Intents are also used to start activities. Intents explicitly start an activity by specifying the class name of the activity and implicitly start an activity by specifying an action and data on which to act upon. When there are number of activities that can perform a certain action on a data, Android uses intent filters to choose the right activity. Intent filters declared by an application register activities, services and broadcast receivers that can perform a certain action on a data. Another common use of intents is to broadcast events across the system.

**Broadcast Receivers**

Broadcast receivers listen for broadcast intents. They are declared in the manifest file of an application along with intent filter that specifies the intents the receiver is listening to. When an application declares broadcast receivers in manifest, the application need not be running in the foreground to receive the events since the system takes care of starting the application when a matching intent is broadcast.

**Services**

Service is a component that performs long running operation in the background without a UI. When an application needs to run for a long time but doesn’t require much user interaction, the application can be implemented as service. As services execute in
the main thread of an application process, Android supports classes that can be used to run time consuming tasks like network lookups in a background thread.

**Files and Databases**

Android provides few data persistence techniques for storing application data. Shared Preferences is a lightweight mechanism for storing primitive data in a map of key/value pairs. It is a preferred means for storing UI state and user preferences. Android provides SQLite relational database library for storing complex application data. It also includes content providers that provide an interface for accessing structured sets of data. Third party applications access database of native applications through content providers.

**Resource Files**

Android supports externalizing resources to manage them efficiently. This helps to define different resource values for different languages, countries, screens and hardware configurations. Android dynamically chooses the right resource file at run time.

### 3.2 Development Environment

The development environment consists of Windows 7 OS, Eclipse Java IDE with Android plug-ins, Sun Java SE Development Kit, Android 2.3.6 operating system (Gingerbread), Android framework API level 10 and Android based handset (Motorola Atrix 2).

Android documentation recommends application development using Eclipse Integrated development Environment (IDE). This helps leverage Eclipse’s features for developing an Android application. Android Development Tools (ADT) is a plug-in for Eclipse that incorporates all the required tools for application development in Android.
using Eclipse. The application is developed using eclipse tools including Android SDK and virtual device manager, Android Emulator and Dalvik Debug Monitor Server (DDMS). Android SDK and virtual device manager manages Android SDKs and virtual devices. Android Emulator emulates a real device and it helps in designing and testing an application. DDMS helps in debugging Android applications.

To install and an android application in a device, the application has to be compiled and packaged to a .apk file that will include all the details about the application as shown in Figure 2 [5]. The next step after packaging is signing the application so that the Android system can identify the author of an application and establish trust relationships between applications. When developing with Eclipse, the IDE takes care of the signing process that is required to install an application in an emulator or a device. Android Debug Bridge (ADB) in Figure 2 is a tool used for communicating between a development machine and an emulator or device.

3.3 Location Technologies

Location APIs in Android use Global Positioning System (GPS) and Network Location Provider to acquire location information. GPS, a satellite based navigation
system provides location information to Android devices that come with built in GPS receivers. Network Location Provider on the other hand uses cell tower and Wi-Fi signals to provide location information. While GPS provides accurate location information, it consumes a lot of battery power and works only outdoors. Network Location Provider, on the other hand consumes less battery power and works both indoors and outdoors, but lacks accuracy. These technologies differ in accuracy, monetary cost and power consumption. Location aware applications can use either of these technologies or both based on their requirements. GPS and Network Provider can be enabled or disabled using location settings in Android devices.

Android also provides Google Maps external library that can be used by location applications to include mapping capabilities in the application.

The location alert feature of the application can be tested using both emulator and real device. The application uses ‘Emulator Control Panel’ in Eclipse to send mock GPS coordinates to the virtual device. Random GPS coordinates are sent to the virtual device to initialize the current user location. After initializing the current location, coordinates of point of interest (POI) are sent to the virtual device. This simulates the situation where the user enters his/her point of interest from outside and triggers the alert.

Android provides a technology called geocoding for translating street address to latitude and longitude coordinates. Since Android APIs responsible for location alerts accept addresses in the form of latitude and longitude coordinates, the application uses geocoding technology to convert street addresses to latitude and longitude coordinates. Geocoding cannot be simulated in the emulator used - API level 10, Platform 2.3.3,
Target Google APIs. It is a known issue and hence real device has to be used for testing geocoding functionality.
Chapter 4

METHODOLOGY

This chapter explains the features of the application and the methodologies used in implementing the features.

4.1 Features of the Application

Add, Edit and Delete Alerts

The application provides alerts for the locations provided by the user. The application allows user to add new alerts and edit and delete existing alerts.

Set Weekly Reminders

The user can choose to receive alerts on specific days of the week. On the specified day, when the user is in the proximity of his/her point of interest, the application beeps to remind the user to complete his/her task. If weekly reminders are not set, application alerts the user whenever he/she approaches the set address.

Maintain List of Favorite Places

The application helps users maintain a list of his/her favorite places and set alerts for these places. Users can add new places and edit and delete existing places in the list.

Location Input Options

Users can manually enter an address of a location or choose an existing address from contacts-list or favorite places list for setting alerts.

Set Radius for Triggering the Alert

The application lets the user specify central location of a region and a radius around it and alerts the user when he/she enters the circular region. The application
displays a list of radius values ranging from 50 m to 3000 m from which the user can select one based on his/her requirements.

**Turn On/Off Alert**

Users can enable or disable alerts as needed. Turning off alerts when they are not required saves battery life.

### 4.2 Methodology

This section explains the steps involved in setting proximity alerts [Figure 3] and the various methodologies used in implementing the features of the application.

**Address Translation**

While users provide street addresses for setting location alerts, the Android API for setting location alerts, accepts addresses in the form of latitude and longitude coordinates. Since addresses provided by users are not in the format expected by Android API, there is a need to convert street addresses to latitude and longitude coordinates for setting proximity alerts. Android uses geocoding technology for this address translation. Geocoding requires a backend service and internet connection to perform address lookups. Forward geocoding converts a street address to latitude and longitude coordinates while reverse geocoding converts latitude and longitude coordinates to a street address. For a given named location, forward geocoding returns a list of matching addresses. The returned address objects are populated with as much details as possible including latitude, longitude, street number, house number, city, country and phone number.
Figure 3. Methodology for Setting Proximity Alerts
Since these lookups are done synchronously, the application implements geocoding in a background thread to prevent the blocking of main UI thread when network connections are slow.

**Location Alerts**

The next step after translating the address into latitude and longitude coordinates is to set alert for the address. Android provides an easy to use API `addProximityAlert` of `LocationManager` class for setting proximity alerts. The API notifies the application by broadcasting an event as user approaches his/her point of interest. The API can use any of the location providers for tracking user location based on location settings in the phone. When both GPS and network provider are enabled in location settings, the API decides which provider to use and user does not have control over it. Section 5.2 explains the API in more detail. Both emulator and real device can be used for testing proximity alerts.

**Broadcast Event Handler**

Once the alert is set using the above API, the location manager takes in charge of the alert and tracks user location. When it detects that the user has crossed the specified radius of his/her point of interest, it fires an intent and the application takes action in response to the intent. The application should register a broadcast receiver that listens for the broadcast intent. This is declared in the manifest file of the application along with intent filter that specifies the intents the receiver is listening to. Since the broadcast receiver is registered in the manifest, the application need not be running in the foreground to receive the broadcast events. The system takes care of starting the application when a matching intent is broadcast.
The location alert application being developed notifies the user on receiving the broadcast intent. It utilizes notification service provided by Android system. The capabilities of notification service include creation of status bar icons, displaying information in the extended status bar window, flashing LEDs, vibrating phone and creating audible alerts.

**Persistence**

All location alerts are saved in the file system so that a list of previously created alerts can be seen by the user whenever he/she opens the application. The application uses Shared Preferences mechanism for data persistence. Shared Preferences is a lightweight mechanism for storing primitive data types in a map of key/value pairs. It is the preferred means for storing UI state and user preferences. When user creates a new alert, the application creates an alert entity object that has all the details about the alert including alert name, address, latitude, longitude, radius, alert id, days to repeat and status of the alert indicating whether the alert is enabled or not. After setting proximity alert for the alert entity, it is persisted in the default shared preferences file of the application. Since shared preferences file can only store primitive data types, the alert entity object needs to be converted to a primitive type in order to save it. The first step is to add the alert entity object to the array list of previously created alerts that is maintained by the application and the next step is to convert the entire array list to a string using object serialization techniques. Since string is a primitive data type, it can be persisted in shared preferences file. Whenever the application is opened, the saved array list of alert
entities is retrieved from shared preferences file using object deserialization technique and displayed to user.

**Edit/Delete Functionality**

The application supports editing and deleting previously created alert entities. When user selects an existing alert, a preference screen (discussed in section 5.8) is opened for editing the alert entity. After user finishes editing, the new data is saved in shared preferences file and existing alert entity is replaced. If user disables the alert, proximity alert corresponding to the alert entity is unregistered.

When user deletes an existing alert, two things are to be taken care of. Proximity alert corresponding to the alert entity should be unregistered from the system and the alert entity should be removed from the shared preferences file so that when user opens the application the next time, deleted alert entity will not be displayed in the alert list.

**Weekly Reminders**

The application supports a feature using which users can choose to receive alerts on specific days of the week. Unless the alert entity is explicitly turned off, user will keep receiving the alert repeatedly every week on the selected days. The application uses an efficient algorithm for implementing this feature that preserves battery life. A time alarm is set for 12 AM on the chosen days and when the alarm event is broadcast at 12 AM on each of the chosen days, the application receives the broadcast event and immediately sets a proximity alert for the specified location and sets it to expire by the end of the day. This way the proximity alert is alive only on the chosen days and turned off on other days thus preserving battery life.
Chapter 5

IMPLEMENTATION

This chapter explains the implementation details of the project including the Android APIs and third party APIs used by the application and the GUI screens of the application. A use case diagram (Figure 4) is provided to give an overview on the implementation of the application.

5.1 Geocoding

As explained earlier in section 4.2, the application uses geocoding technology to convert street addresses provided by user to latitude and longitude coordinates. As geocoding lookups are done on the server, the application requires internet access to implement this functionality. The permission constant INTERNET is declared in manifest file as shown below to provide internet access to the application.

\[<uses-permission android:name="android.permission INTERNET"/>\]

Android system provides a class named Geocoder that includes all APIs for implementing geocoding. User can specify the locale to be used using the constructor of this class. Locale specifies country and language details to be used by the APIs of the class while presenting information to users. When no locale specific details are provided, the default system locale is used. The API getFromLocationName when queried returns a list of addresses corresponding to user specified address. When user specified address is not meaningful or network is unavailable, the API might not return a valid address. It might throw an exception or return an empty list of addresses.
The network lookups for performing geocoding translations are done synchronously and will block the calling thread when network connections are slow. When main UI thread is the calling thread, Android might open a force close dialog to keep the UI responsive. To avoid this situation, the application implements geocoding in a separate background thread. Android includes a class named AsyncTask that provides an easy mechanism for moving operations like network lookups to a background thread and publishing results of the operation on the primary UI thread. The application has a class named ForwardGeocoder that subclasses AsyncTask and overrides its methods to implement geocoding in a background thread. Complete implementation details can be seen in the file SetAlertActivity.java in the appendix. The method doInBackground of AsyncTask class performs time consuming operations in the background thread while method onPostExecute invoked on the UI thread publishes the result of the computation.

5.2 Proximity Alerts

Android includes LocationManager class to provide location services supported by the device to an application. The application uses API addProximityAlert of this class for setting proximity alert for a given address. The API requires latitude/longitude geographic coordinates, radius around the central point of alert region and an expiration time for the alert. It notifies the application when user crosses the specified radius of the location and takes care of deleting the alert after the expiry time. It also decides the frequency in which location updates are received and automatically chooses a location provider based on location settings. The API also accepts a PendingIntent parameter that
gets fired when the alert is triggered. A PendingIntent object wraps an intent that gets fired at a later point of time. The API usage is shown below.

```java
locationManager.addProximityAlert(latitude, longitude, radius, expiration,
pendingIntent);
```

Full implementation details can be seen in the file ProximityAlertManager.java in appendix. The application declares the following permissions in the manifest to make use of location providers.

```xml
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

GPS requires fine permission while network provider requires coarse permission. An application that has been granted fine permission will be granted coarse permission implicitly. So, it is not necessary to declare ACCESS_COARSE_LOCATION when ACCESS_FINE_LOCATION is declared.

The location providers GPS and Network provider should be enabled in phone location settings for receiving proximity alerts. When both the providers are enabled, the API addProximityAlert chooses one of the location providers depending on the user’s location with respect to the set destination. As mentioned earlier, GPS provides accurate location information but consumes more battery power and Network provider consumes less battery power but lacks in accuracy.
Figure 4. A UML Use Case Diagram
5.3 Persistence

Whenever the application is opened, it needs to display the list of alerts previously created by the user so that the user can view, edit or delete the alerts. The application stores the list of previously created alerts in permanent storage in order to display the list whenever it is opened.

When user creates a new alert, the application creates an alert entity object that has all the details about the alert including alert name, address, latitude, longitude, radius, alert id, days to repeat and status of the alert indicating whether the alert is enabled or not. The created alert entity object is added to an array list which is then persisted in a file using shared preferences mechanism.

Shared preferences saves data in the form of <key, value> pairs in a map. Shared preferences are shared by all the components of the application but are not available to other applications. The application uses the API getDefaultSharedPreferences in PreferenceManager class to get the default shared preferences file and uses that file to save the array list of alert entities. As explained in section 4.2, the application needs to serialize the array list to a string in order to save it in shared preferences file. It makes use of ObjectSerializer class provided by Apache Pig platform to accomplish this. This class has functionalities to serialize the array list of alert entities to a string and to deserialize the string back to an array list of alert entities. The serialized array list in the form of string is then stored in default shared preferences file. Android provides a class named SharedPreferences.Editor for editing shared preferences file. The application uses putString method of this class to persist serialized array list in string format to shared
preferences file and uses getString method of SharedPreferences class to retrieve the serialized array list. Further details can be found in source files in the appendix.

5.4 Broadcast Event

When a proximity alert is set, location manager keeps tracking user location in the background. When user crosses the set radius boundary of his/her point of interest, location manager fires the registered pending intent (explained in section 5.2) with extra information on whether user is entering or exiting the set radius boundary. The application specifies its interest in the broadcast intent by registering a broadcast receiver in its manifest file using intent filters. When broadcast receivers are registered in the manifest, the application is not required to run in the foreground to receive the proximity alerts. When an alert is triggered, the onReceive method of the registered broadcast receiver gets called by the system. Inside this method, the application checks the extra information sent by location manager and notifies the user only when user enters the radius boundary of the set location. Implementation details can be seen in file PendingAlertReceiver.java in the appendix.

5.5 Notifications

The application makes use of Android notification service to notify user when an alert is triggered. Notification and NotificationManager classes in Android provide services to notify user by creating status bar icons, displaying messages in extended status bar window, flashing LEDs, vibrating phone and making audible alerts. API usage can be seen in file PendingAlertRecevier.java in the appendix.
5.6 Address Input Options

The application provides several options to enter address for setting proximity alerts. User can type an address, choose an existing address from native contacts application or choose an existing address from a list containing his/her favorite places. The application displays a menu with these three choices from which user can select one. The next two sections discuss the implementation details of these options.

Interaction with Contact-List

Android includes content providers, a mechanism to share data between applications. Using content providers, applications can publish and consume data based on an URI addressing model. The contact manager application native to Android system exposes its database to third party applications using content providers. Applications require READ_CONTACTS permission to access contact-list. In order to pop up the contact-list from the application, it defines the permission in manifest file as shown below.

```xml
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

android.provider package includes a class called ContactsContract that has all the APIs to retrieve and modify contacts details. The contacts contract content provider refers to a database that has three main tables Data, RawContacts and Contacts to store contact details. The application uses ContactsContract.Contacts table to retrieve address information of the contacts. It starts contacts-list activity as shown below

```java
Intent intent = new Intent(Intent.ACTION_PICK,
ContactsContract.Contacts.CONTENT_URI);
```
ACTION_PICK shown in the code snippet is a predefined action that picks an item from the given data URI CONTENT_URI and returns the selected item. PICKCONTACT is the request code that will be returned in onActivityResult when contacts-list activity is exited. When a contact is chosen in the contacts list, the detail of the chosen contact is returned to the calling activity through onActivityResult. The application then extracts the address details of the contact to set proximity alert for the contact address. If the selected contact does not have an address associated with it, a dialog box is shown to the user with a message saying that there is no address associated with the contact. Implementation details can be seen in file SetAlertActivity.java in the appendix.

**Favorite-Places List**

The application allows the user to create a list of his/her favorite places, add new places to the list and modify existing places in the list. The application stores the list in shared preferences file so as to display the list whenever the application is opened. User can select an address from the favorite places list and set proximity alert for the address.

Similar to alert list explained in section 5.3, the favorite-places list is serialized to a string and stored in shared preferences file. The application deserializes the string to an array list of favorite places to display it to user.

When user chooses favorite places list option for setting proximity alert, the application displays the list of previously persisted places from which the user can pick one. When user picks a place from the list, address of the place is returned to the calling
activity. The calling activity processes the returned address and sets proximity alert for the address.

5.7 Weekly Reminders Feature

The application supports a feature using which users can choose to receive alerts on specific days of the week. Unless explicitly turned off, the user will keep receiving the alert repeatedly every week on the selected days. The application uses the following steps to implement this feature.

a) The application sets a time alarm for 12 AM that wakes up the application on each of the user selected days. This alarm wakes up just the application and not the end user. Android includes an AlarmManager class that the application uses for setting time alarms. The setRepeating API of this class enables the application to set repeated time alarms that wakes up the application at 12 AM on the chosen days every week. Similar to proximity alerts, the application registers a broadcast receiver that is invoked by the system when the time alarm is triggered.

b) When alarm manager fires the registered pending intent, onReceive method of the broadcast receiver gets called and it sets a proximity alert with an expiration time of 24 hours. The latitude and longitude coordinates for setting the proximity alert are extracted from extras attached to the received intent as shown below.

   ```java
   public void onReceive(Context context, Intent intent) {
       proximityAlertManager = new ProximityAlertManager(context);
       long MILLISECS_IN_DAY = 86400000;
   }
   ```
int alertId = intent.getIntExtra(context.getString(R.string.request_code), -1);

double latitude = intent.getDoubleExtra("lat", 91);

double longitude = intent.getDoubleExtra("long", 181);

String radius = intent.getStringExtra("radius");

proximityAlertManager.setProximityAlert(latitude, longitude, radius, MILLISECS_IN_DAY, alertId);

c) When location manager triggers the alert on the chosen day by firing the pending intent, the application receives the intent using its broadcast receiver and notifies the user.

Thus, the user is notified of the proximity alert only on the chosen days when user reaches the set destination. When user chooses multiple days of the week, the application turns on the proximity alert on each of the chosen days and turns it off by the end of the days. Since alerts are alive only on user chosen days, this methodology for weekly reminders saves battery life.

5.8 GUI Screens

This section shows the different screens used by the application and discusses the user interface controls used in designing the screens of the application.

Alert-List Screen

The screen shown in Figure 5 is alert-list screen that displays the list of location alerts set by the user. It is the main screen that is displayed when the application is opened. The application subclasses ListActivity class of Android that has functionalities to display list of items from a data source like array or cursor.
The application stores the list of alerts in an array list in a shared preferences file and binds the ListActivity to this array using ArrayAdapter class. Complete implementation details can be seen in file LocationAlertActivity.java in appendix.

As shown in the figure, the screen includes an options menu that has two items ‘Add Alert’ and ‘About’. The first item is to add a new alert and the second item is to provide information about the application. The application overrides onCreateOptionsMenu API in Activity class and inflates the options menu defined in XML resource file in the method using MenuInflator APIs. The onOptionsItemSelected() method in Activity class is overridden to include functionalities
that are invoked when an item in the options menu is clicked. The alert-list screen also has a context menu that is shown when an alert item in the alert list is long pressed. The context menu has ‘Delete’ item which when clicked removes the alert from the alert list and cancels the corresponding proximity alert in location manager. Similar to options menu, the APIs onCreateContextMenu and onContextItemSelected in Activity class provide functionalities for working with context menu.

Figure 6. Edit-Alert Screen
Edit-Alert Screen

The edit-alert screen shown in Figure 6 is displayed when user clicks on an alert in alert-list screen for editing the alert item. The activity corresponding to this screen uses preferences API that provides system style user interface screen to the application.

![Image of edit-alert screen]

*Figure 7. Location Input Options*

This enhances user experience as this screen is consistent with the screens used by native and other third party applications. The screen elements are called preferences and they are declared in an XML file. The main preferences provided by this API are list preference, checkbox preference, edit text preference and ringtone preference.
When preferences screen is used, the system takes care of persisting the settings of the preference screen in the default shared preferences file of the application. The system saves the default preferences file of the application in data folder of the file system. Android provides PreferenceActivity class to host application preference screen. The activity for edit-alert screen subclasses PreferenceActivity and it has functionalities for creating new alerts and editing existing alerts.

Figure 8. Edit-Address Dialog
The alert-list activity that launches edit-alert activity sends an intent that specifies whether the user wants to create a new alert or edit an existing alert. User clicks on ‘Add alert’ options menu item in alert-list screen for creating a new alert and clicks on an alert item in the alert-list screen to edit an existing item.

By default, preference screens display the last set preferences. The edit-alert activity overrides this behavior by clearing the preference screen for new alerts and populating the preference screen with the respective alert entity data for existing alerts.
The edit-alert screen has edit text preference for entering alert name, a list preference for providing radius values and a check box preference for enabling and disabling alerts. The preference element for address opens up a dialog which provides options to enter address (Figure 7) and the preference element for weekly reminders displays a dialog that shows a radio button list (Figure 10) containing days of week. The next section explains the dialogs and activities that are launched from edit-alert screen.

**Screens Launched from Edit-Alert Screen**

The application opens up a dialog shown in Figure 7 when location address preference (Figure 6) is clicked. The dialog displays different choices to enter address.
The first item ‘Use Contacts’ opens up native contacts list when clicked. By clicking on a contact in the contacts list, user can set location alert for the contact address. If no address is stored for the contact, an alert dialog is opened to inform the user that the contact has no associated address. The third item ‘Enter Address’ when selected opens up an alert dialog with textbox control. User can type an address in the box for setting proximity alert. The screen corresponding to this dialog can be seen in Figure 8. The second item ‘Use Favorite Places List’ opens up a new activity using which the user can create a list of his favorite places and select an address from the list.
When ‘Alert Radius’ preference is chosen, a list preference dialog box with radio button list as shown in Figure 9 is displayed with radius values ranging from 50 m to 3000 m from which user can choose one for setting the alert.

A check box preference is used for enabling and disabling alerts. User can turn on or turn off alerts when needed using this preference. When existing alerts that are enabled are turned off, the corresponding proximity alerts are unregistered from location manager. When ‘Repeat’ preference is clicked in Figure 6, an alert dialog with list of days is opened. The list items include a check box with them that enable user to choose multiple days of the week for the same proximity alert location. By clicking the
ok button in the dialog shown in Figure 10 user can set proximity alerts that get repeated every week on the selected days.

When user clicks on the ‘Save’ button in edit-alert screen after setting values for all the preferences, the activity executes geocoding in a background thread and checks if the address entered by user points to a valid location. If address entered by the user is invalid, a dialog box is shown that prompts the user to enter valid address.

**Favorite-Places-List Screens**

As mentioned earlier, the application lets user maintain a list of his favorite places and choose an address from the list for setting location alerts. Similar to alert list, the favorite places list is persisted in shared preferences file and displayed to user whenever the application is opened. The application also has functionalities to add new places in favorite-places list, edit and delete existing places in the list. These screens are shown in Figure 11 and Figure 12.
Chapter 6
FINDINGS, INTERPRETATIONS AND FUTURE WORK

This chapter discusses some of the problems that were encountered during the development of the application and the solutions devised to solve them, then analyses some of the performance issues and also discusses few enhancements that can be added to the application in the future.

While designing android applications, there is always a question of whether to use database or shared preferences technology for persistence. Shared preferences technology was chosen mainly because the data to be saved by the application is relatively smaller. Shared preferences is a light weight mechanism for persistence and data retrieval is easier and quicker with this mechanism when compared to database. The application avoids maintaining large number of keys in shared preferences file by adding all alert entities to an array list and persisting the entire array list in the shared preferences file using a single key.

One of the issues faced when developing the application relates to the radius value that was chosen for setting proximity alerts. The application missed to alert the user under certain situations. For instance, when user provided the name of a place like “CSU, Sacramento” for address and provided a radius value of say 200 m, the application failed to alert and notify the user when he/she entered the premises of the university. The reason was found to be with the chosen radius value. Geocoding translates the given address to latitude, longitude coordinate pair that represents some location in the university and a radius of 200 m around that point is not large enough to cover all the area of the
university. This problem can be avoided by choosing a larger radius value for places like this so that the application will not miss to alert the user when he is in certain location of the university. One of the other issues was that the application was notifying the user only when it ran in the foreground and failed to alert the user otherwise. This issue was avoided by registering the broadcast receiver in the manifest instead of registering it in a source code file. Android system wakes up the application even if it is not in the foreground when the required broadcast receivers are registered in the manifest file.

Another issue faced relates to geocoding. Geocoding did not work in the emulator version used and it was found that it was a known issue with the system for that particular version. Details about this are included in section 3.3

As discussed in previous chapters, the application uses both the location providers GPS and Network Provider for triggering proximity alerts. Using GPS for tracking user location has an adverse impact on battery life. One of the steps taken by the application towards preserving battery life is to use an efficient methodology for implementing weekly reminders feature as discussed in section 5.7. The method turns on the alerts only on the days when user wants to be reminded and turns them off on other days of the week. For other regular reminders that alert the user on any day of the week when he/she reaches the specified destination, it is in the hands of the Android API to decide on the location provider. User can also make use of turn on/off feature provided by the application to save battery life. Alerts can be turned on when required and turned off at other times. The default implementation provided by Android system for setting proximity alerts sets the minimum time interval (in milliseconds) between location
updates and minimum distance (in meters) between location updates to a value of one [2]. This might consume a lot of battery power when alerts have long expiration time and application has no control over it. Another drawback with the default implementation is that an application will need permission android.permission.ACCESS_FINE_LOCATION in manifest for accessing GPS even if the application only requires coarse location information from network provider [2].

Apart from the issues discussed above, the application was tested to work seamlessly with no significant adverse impact on battery life. The application did not freeze under any condition and always alerted the user with notifications.

A part of future work, the application can implement an algorithm that gives a better battery life when compared to the default implementation provided by Android. The algorithm should reduce the frequency of location updates received and should be capable of switching between location providers to balance accuracy and power consumption. Google maps can be added as one of the address input options since it is easier to find a location using maps when address of a place is not known.
APPENDIX

Source Code

AlertEntity.java

package com.sacstate.csc502.locationalert;

import java.io.Serializable;

/**
 * @author Sukanya.
 * This class represents a proximity alert entity.
 */
public class AlertEntity implements Serializable {

    private static final long serialVersionUID = 1L;
    String name;
    String address;
    Boolean enabled;
    double latitude;
    double longitude;
    String radius;
    Boolean isRepeatTurnedOn;
    int[] alertIds;
    boolean[] repeatOnDays;

    /**
     * Constructor
     */
    public AlertEntity(String name, String address, Boolean enabled,
                        int[] alertIds, double latitude, double longitude, String radius,
                        boolean isRepeatTurnedOn, boolean[] repeatOnDays) {
        this.name = name;
        this.enabled = enabled;
        this.address = address;
        this.alertIds = alertIds;
        this.latitude = latitude;
        this.longitude = longitude;
        this.radius = radius;
        this.isRepeatTurnedOn = isRepeatTurnedOn;
        this.repeatOnDays = repeatOnDays;
    }
*/ (non-Javadoc)
* @see java.lang.Object#toString()
*/
@Override
public String toString() {
    return name;
}
}
AlertListManager.java

package com.sacstate.csc502.locationalert;

import java.io.IOException;
import java.util.ArrayList;
import android.content.Context;
import android.content.SharedPreferences;
import android.widget.ListView;

/**<p>
* @author Sukanya.
* Class responsible for managing the list of alerts created by user.
*/

public class AlertListManager {

    static ListView alertListView;
    Context context;
    SharedPreferences defaultSharedPreferences;

    /**
     * Constructor
     */
    public AlertListManager(Context context, ListView listView) {
        this.context = context;
        alertListView = listView;
        defaultSharedPreferences = SharedPreferencesProvider.getDefaultSharedPreferences(context);
    }

    public AlertEntity createAlertEntityFromDefaultSharedPreferences(int[] alertIds, double latitude, double longitude, boolean isRepeatTurnedOn, boolean[] repeatOnDays) {
        boolean alertEnabled = defaultSharedPreferences.getBoolean(context.getString(R.string.preference_turnon_key), false);
        String alertName = defaultSharedPreferences.getString(context.getString(R.string.preference_alertname_key), "");
        String address = defaultSharedPreferences.getString(context.getString(R.string.preference_address_key), "");
        String radius = defaultSharedPreferences.getString(context.getString(R.string.preference_radius_key), "1000");
    }
}
if (alertName.isEmpty())
    alertName = "(No Name)");
    return new AlertEntity(alertName, address, alertEnabled,
                        alertIds,
                        latitude, longitude, radius, isRepeatTurnedOn,
                        repeatOnDays);

public void addAlertEntityToAlertList(ArrayList<AlertEntity> alertList,
                              AlertEntity alertEntity) {
    alertList.add(alertEntity);
}

public AlertEntity removeAlertEntityFromAlertList(
                              ArrayList<AlertEntity> alertList, int index) {
    return alertList.remove(index);
}

public void saveAlertListToDefaultSharedPreferences(
                              ArrayList<AlertEntity> alertList) {
    String serializedObj = null;
    try {
        serializedObj = ObjectSerializer.serialize(alertList);
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    SharedPreferences.Editor editor = defaultSharedPreferences.edit();
    editor.putString(context.getString(R.string.alertlist_key),
                        serializedObj);
    editor.commit();
}

public ArrayList<AlertEntity>
getAlertListFromDefaultSharedPreferences() {
    String serializedAlertList = defaultSharedPreferences.getString(
                                    context.getString(R.string.alertlist_key),
                                        null);
    ArrayList<AlertEntity> alertList = null;

    if (serializedAlertList != null) {
        try {
            alertList = (ArrayList<AlertEntity>) ObjectSerializer
                                .deserialize(serializedAlertList);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
    return alertList;
} catch (IOException e) {
    e.printStackTrace();
}

} else
    alertList = new ArrayList<AlertEntity>();
    return alertList;
}

public static AlertEntity getAlertEntityAtListPos(int listItemPos) {
    return (AlertEntity) alertListView.getItemAtPosition(listItemPos);
}

public static AlertEntity getAlertEntityForAlertId(int id, ArrayList<AlertEntity> alertList) {
    for (AlertEntity alertEntity : alertList) {
        for (int alertId : alertEntity.alertIds) {
            if (id == alertId)
                return alertEntity;
        }
    }
    return null;
}
}
FavoritePlaceEntity.java

package com.sacstate.csc502.locationalert;

import java.io.Serializable;

/**
 * @author Sukanya. This class represents a favorite place entity created by user
 */
public class FavoritePlaceEntity implements Serializable {

    private static final long serialVersionUID = 1L;
    String name;
    String address;

    /**
     * @param name, Name of the favorite place
     * @param address, Address of the favorite place
     */
    public FavoritePlaceEntity(String name, String address) {
        this.name = name;
        this.address = address;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return name;
    }
}
**FavoritePlacesListActivity.java**

```java
package com.sacstate.csc502.locationalert;

import java.util.ArrayList;
import android.app.ListActivity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

/**
 * @author Sukanya.
 * Activity that displays the list of favorite places created
 * by user. Clicking on an item in the list opens a new activity for
 * editing the item. The screen also supports an options menu which
 * helps in adding new places to the favorite place list.
 */
public class FavoritePlacesListActivity extends ListActivity {

    ArrayList<FavoritePlaceEntity> favoritePlacesList;
    static final int SET_FAVORITE_PLACE_ACTIVITY = 4;
    SharedPreferences defaultsharedPref;
    FavoritePlacesListManager favoritePlacesListManager;
    ArrayAdapter<FavoritePlaceEntity> arrayAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        defaultsharedPref = SharedPreferencesProvider.getDefaultSharedPreferences(this);
        favoritePlacesListManager = new FavoritePlacesListManager(this, getListView());
        favoritePlacesList = favoritePlacesListManager.getFavoritePlaceList();
        arrayAdapter = new ArrayAdapter<FavoritePlaceEntity>(this,
```
android.R.layout.simple_list_item_1, favoritePlacesList);
setListAdapter(arrayAdapter);
registerForContextMenu(getListView());
}

protected void onListItemClick(ListView l, View v, int position, long id) {
    Intent calleeIntent = new Intent();
    FavoritePlaceEntity favoritePlaceEntity = FavoritePlacesListManager
        .getFavoritePlaceEntityAtListPos(position);
    calleeIntent.putExtra(getString(R.string.favorite_place),
        favoritePlaceEntity.address);
    setResult(RESULT_OK, calleeIntent);
    finish();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.favoriteplace_list_options_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.add_place:
        Intent intent = new Intent(this, SetFavoritePlaceActivity.class);
        intent.putExtra(getString(R.string.new_favoriteplace_item), true);
        startActivityForResult(intent, SET_FAVORITE_PLACE_ACTIVITY);
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.favoriteplace_list_context_menu, menu);
}

public boolean onContextItemSelected(MenuItem item) {
AdapterContextMenuInfo info = (AdapterContextMenuInfo) item
    .getMenuInfo();
switch (item.getItemId()) {
case R.id.edit_place:
    Intent intent = new Intent(this, SetFavoritePlaceActivity.class);
    intent.putExtra(getString(R.string.new_favoriteplace_item), false);
    intent.putExtra(getString(R.string.listitem_pos),
        (int) info.position);
    startActivityForResult(intent, SET_FAVORITE_PLACE_ACTIVITY);
    return true;
case R.id.delete_place:
    FavoritePlaceEntity favoritePlaceEntity =
        favoritePlacesListManager
            .removeFavoritePlaceEntityFromFavoritePlaceList(
                favoritePlacesList, (int) info.position);
    arrayAdapter.notifyDataSetChanged();
    favoritePlacesListManager
        .saveFavoritePlaceListTodefaultSharedPreferences(favoritePlacesList);
    return true;

default:
    return super.onContextItemSelected(item);
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
    case SET_FAVORITE_PLACE_ACTIVITY:
        if (resultCode == RESULT_OK && data != null) {
            FavoritePlaceEntity favoritePlaceEntity = null;

            if (data.getBooleanExtra(
                getString(R.string.new_favoriteplace_item),
                false)) {
                favoritePlaceEntity = favoritePlacesListManager
                    .createFavoritePlaceEntityFromDefaultSharedPreferences();
            favoritePlacesListManager
                .addFavoritePlaceEntityToFavoritePlacesList(
                    favoritePlacesList,
                    favoritePlaceEntity);
            } else {
                int listItemPos = data.getIntExtra(
                    getString(R.string.listitem_pos), -1);
            }
        }
    }
}
if (listItemPos != -1) {
    favoritePlaceEntity =
    favoritePlacesListManager
        .createFavoritePlaceEntityFromDefaultSharedPreferences();
    favoritePlacesList
        .set(listItemPos, favoritePlaceEntity);

    favoritePlacesListManager
        .saveFavoritePlaceListToDefaultSharedPreferences(favoritePlacesList);
}

@Override
protected void onResume() {
    super.onResume();
    arrayAdapter.notifyDataSetChanged();
}
}
**FavoritePlacesListManager.java**

```java
package com.sacstate.csc502.locationalert;

import java.io.IOException;
import java.util.ArrayList;
import android.content.Context;
import android.content.SharedPreferences;
import android.widget.ListView;

/**<*
 * @author Sukanya. Class responsible for managing the list of favorite places created by user.
 */
public class FavoritePlacesListManager {

    static ListView favoritePlacesListView;
    Context context;
    SharedPreferences defaultSharedPreferences;

    public FavoritePlacesListManager(Context context, ListView listView) {
        this.context = context;
        favoritePlacesListView = listView;
        defaultSharedPreferences = SharedPreferencesProvider.getDefaultSharedPreferences(context);
    }

    public FavoritePlaceEntity createFavoritePlaceEntityFromDefaultSharedPreferences() {
        String favoritePlaceName = defaultSharedPreferences.getString(
            context.getString(R.string.preference_favoriteplace_name_key), "");
        String address = defaultSharedPreferences.getString(context
            .getString(R.string.preference_favoriteplace_address_key), "");

        if (favoritePlaceName.isEmpty())
            favoritePlaceName = "(No Name)";

        return new FavoritePlaceEntity(favoritePlaceName, address);
    }

    public void addFavoritePlaceEntityToFavoritePlacesList(
```
```
ArrayList<FavoritePlaceEntity> favoritePlaceList,
FavoritePlaceEntity favoritePlaceEntity) {
    favoritePlaceList.add(favoritePlaceEntity);
}

public FavoritePlaceEntity removeFavoritePlaceEntityFromFavoritePlaceList(
    ArrayList<FavoritePlaceEntity> favoritePlaceList, int index) {
    return favoritePlaceList.remove(index);
}

public void saveFavoritePlaceListToDefaultSharedPreferences(
    ArrayList<FavoritePlaceEntity> favoritePlaceList) {
    String serializedObj = null;
    try {
        serializedObj = ObjectSerializer.serialize(favoritePlaceList);
    } catch (IOException e1) {
        e1.printStackTrace();
    }
    SharedPreferences.Editor editor = defaultSharedPreferences.edit();
    editor.putString(context.getString(R.string.favoriteplacelist_key), serializedObj);
    editor.commit();
}

public ArrayList<FavoritePlaceEntity>
getFavoritePlaceListFromDefaultSharedPreferences() {
    String serializedList = defaultSharedPreferences.getString(
        context.getString(R.string.favoriteplacelist_key), null);
    ArrayList<FavoritePlaceEntity> favoritePlaceList = null;
    if (serializedList != null) {
        try {
            favoritePlaceList = (ArrayList<FavoritePlaceEntity>)
                ObjectSerializer.deserialize(serializedList);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else
        favoritePlaceList = new ArrayList<FavoritePlaceEntity>();
    return favoritePlaceList;
```
public static FavoritePlaceEntity getFavoritePlaceEntityAtListPos(
        int listItemPos) {
    return (FavoritePlaceEntity) favoritePlacesListView
            .getItemAtPosition(listItemPos);
}
}
LocationAlertActivity.java

package com.sacstate.csc502.locationalert;

import java.util.ArrayList;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.AdapterContextMenuInfo;

/**
 * @author Sukanya. Represents the main activity that is displayed when the
 * application is opened. It displays the list of proximity alerts
 * created by user. It also has menus to add, edit and delete alerts.
 */
public class LocationAlertActivity extends ListActivity {

    ArrayList<AlertEntity> alertList;
    static final int SET_ALERT_ACTIVITY = 1;
    SharedPreferences defaultsharedPref;
    ArrayAdapter<AlertEntity> arrayAdapter;
    ProximityAlertManager proximityAlertManager;
    RepeatingAlarmManager repeatingAlarmManager;
    static final int DIALOG_ABOUT_ID = 0;
    static final int DIALOG_NO_LOCATIONPROVIDER_ID = 1;
    AlertListManager alertListManager;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        defaultsharedPref = SharedPreferencesProvider
    }
}
getSharedPreferences(this);
proximityAlertManager = new ProximityAlertManager(this);
repeatingAlarmManager = new RepeatingAlarmManager(this);
alertListManager = new AlertListManager(this, getListView());
alertList = alertListManager.getAlertListFromDefaultSharedPreferences();
arrayAdapter = new ArrayAdapter<AlertEntity>(this, android.R.layout.simple_list_item_1, alertList);
setListAdapter(arrayAdapter);
registerForContextMenu(getListView());
}

protected void onListItemClick(ListView l, View v, int position, long id) {
    Intent intent = new Intent(this, SetAlertActivity.class);
    intent.putExtra(getString(R.string.new_alert), false);
    intent.putExtra(getString(R.string.listitem_pos), position);
    startActivityForResult(intent, SET_ALERT_ACTIVITY);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.add_alert:
        Intent intent = new Intent(this, SetAlertActivity.class);
        intent.putExtra(getString(R.string.new_alert), true);
        startActivityForResult(intent, SET_ALERT_ACTIVITY);
        return true;
    case R.id.about:
        showDialog(DIALOG_ABOUT_ID);
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

case R.id.add_alert:
    Intent intent = new Intent(this, SetAlertActivity.class);
    intent.putExtra(getString(R.string.new_alert), true);
    startActivityForResult(intent, SET_ALERT_ACTIVITY);
    return true;
    
    case R.id.about:
showDialog(DIALOG_ABOUT_ID);
    return true;
    
default:
return super.onOptionsItemSelected(item);
    
    return super.onOptionsItemSelected(item);
}
ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}

public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item
    .getItemInfo();
    switch (item.getItemId()) {
        case R.id.delete_alert:
            AlertEntity alertEntity = alertListManager
            .removeAlertEntityFromAlertList(alertList,
            (int) info, position);
            if (proximityAlertManager.checkForLocationProviders() != null)
                cancelAllAlarms(alertEntity.alertIds);
            arrayAdapter.notifyDataSetChanged();

            alertListManager.saveAlertListToDefaultSharedPreferences(alertList);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case SET_ALERT_ACTIVITY:
            if (resultCode == RESULT_OK && data != null) {
                AlertEntity alertEntity = null;

                double latitude = data.getDoubleExtra(
                    getString(R.string.latitude), 91);
                double longitude = data.getDoubleExtra(
                    getString(R.string.longitude), 181);
                boolean[] repeatOnDays = data
                    .getBooleanArrayExtra(getString(R.string.repeat_on_days));
                boolean isRepeatTurnedOn = data.getBooleanExtra(
                    getString(R.string.is_repeat_turnedon),
                    false);
int[] uniqueIds =
getUniqueIdsForAlertEntity(isRepeatTurnedOn,
repeatOnDays);

if (data.getBooleanExtra(getString(R.string.new_alert),
false)) {
    alertEntity = alertListManager
        .createAlertEntityFromDefaultSharedPreferences(          
uniqueIds, latitude,
longitude,
isRepeatTurnedOn,
repeatOnDays);

    alertListManager.addAlertEntityToAlertList(alertList,
alertEntity);
} else {
    int listItemPos = data.getIntExtra(          
getString(R.string.listitem_pos), -1);
    if (listItemPos != -1) {
        int[] alertIdsOfExisingAlertEntity = alertList
            .get(listItemPos).alertIds;
        cancelAllAlarms(alertIdsOfExisingAlertEntity);
        alertEntity = alertListManager
            .createAlertEntityFromDefaultSharedPreferences(          
uniqueIds,
latitude, longitude,
isRepeatTurne
don, repeatOnDays);
        alertList.set(listItemPos, alertEntity);
    }
}

alertListManager
    .saveAlertListTodefaultSharedPreferences(alertList);

if (alertEntity.enabled) {
    if (proximityAlertManager.checkForLocationProviders() != null) {
        int[] ids = alertEntity.alertIds;
        int indexId = 0;
        if (alertEntity.isRepeatTurnedOn) {
            boolean[] days =
                alertEntity.repeatOnDays;

            for (int i = 0; i < days.length; i++) {
                if (days[i]) {
                    ids[indexId] = uniqueIds[i];
                    indexId++;
                }
            }
        }
    }
}
if (indexId < ids.length)

repeatingAlarmManager.setDayBasedRepeatingAlarm(
  i + 1, ids[indexId++], alertEntity.latitude, alertEntity.longitude, alertEntity.radius);

proximityAlertManager.setProximityAlert(
  alertEntity.latitude, alertEntity.longitude, alertEntity.radius, -1, ids[0]);

showDialog(DIALOG_NO_LOCATIONPROVIDER_ID);

private void cancelAllAlarms(int[] alertIds) {
  for (int i = 0; i < alertIds.length; i++) {
    repeatingAlarmManager.cancelRepeatingAlarm(alertIds[i]); // time alarm
    proximityAlertManager.removeProximityAlert(alertIds[i]); // location alarm
  }
}

@Override
protected void onResume() {
  super.onResume();
  arrayAdapter.notifyDataSetChanged();
}

protected Dialog onCreateDialog(int id) {
  Dialog dialog = null;
  switch (id) {
    case DIALOG_ABOUT_ID:
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setMessage(
      showDialog(DIALOG_NO_LOCATIONPROVIDER_ID);
      } else
    } else
    }

private void cancelAllAlarms(int[] alertIds) {
  for (int i = 0; i < alertIds.length; i++) {
    repeatingAlarmManager.cancelRepeatingAlarm(alertIds[i]); // time alarm
    proximityAlertManager.removeProximityAlert(alertIds[i]); // location alarm
  }
}

@Override
protected void onResume() {
  super.onResume();
  arrayAdapter.notifyDataSetChanged();
}

protected Dialog onCreateDialog(int id) {
  Dialog dialog = null;
  switch (id) {
    case DIALOG_ABOUT_ID:
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setMessage(
"This application provides proximity alerts to users using GPS/Network Provider.")
.setNeutralButton("OK",
    new
    DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int id) {
            dialog.dismiss();
        }
    });
    AlertDialog alert = builder.create();
    return alert;
  case DIALOG_NO_LOCATIONPROVIDER_ID:
    AlertDialog.Builder
    noLocProviderBuilder = new
    AlertDialog.Builder(this);
    noLocProviderBuilder
    .setMessage(
        "Cannot find location providers. Make sure to turn on GPS or Network provider in location settings")
    .setNeutralButton("OK",
        new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,
                int id) {
                dialog.dismiss();
            }
        });
    AlertDialog noLocProviderDialog =
    noLocProviderBuilder.create();
    return noLocProviderDialog;
  default:
    dialog = null;
    }
    return dialog;
}

private int[] getUniqueIdsForAlertEnity(boolean isRepeatTurnedOn,
    boolean[] repeatOnDays) {
    int[] uniqueIds;
    if (isRepeatTurnedOn) {
        int count = 0;
        for (int i = 0; i < repeatOnDays.length; i++) {
            if (repeatOnDays[i])
                count++;
        }
    }
uniqueIds = generateUniqueAlertIds(count);

} else {
    uniqueIds = generateUniqueAlertIds(1);
}
return uniqueIds;
}

private int[] generateUniqueAlertIds(int numberOfIds) {
    int id = 1;
    boolean isEqual = false;
    int[] resultArray = new int[numberOfIds];
    int resultArrayIndex = 0;

    while (numberOfIds != 0) {
        outerLoop: for (AlertEntity alertEntity : alertList) {
            for (int i = 0; i < alertEntity.alertIds.length; i++) {
                if (id == alertEntity.alertIds[i]) {
                    isEqual = true;
                    break outerLoop;
                }
            }
        }

        if (!isEqual) {
            resultArray[resultArrayIndex++] = id;
            numberOfIds--;
        }
        isEqual = false;
        id++;
    }
    return resultArray;
}
ObjectSerializer.java

package com.sacstate.csc502.locationalert;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

/**
 * @author Sukanya. Class responsible for serializing array list of alert
 * entities to a string and deserializing string to an array list of
 * alert entities. It is a slightly modified version of file
 * ObjectSerializer.java provided by Apache pig platform.
 * *
 */
public class ObjectSerializer {

    public static String serialize(Serializable obj) throws IOException {
        ByteArrayOutputStream serialObj = null;
        if (obj == null)
            return "";
        serialObj = new ByteArrayOutputStream();
        ObjectOutputStream objStream = new ObjectOutputStream(serialObj);
        objStream.writeObject(obj);
        objStream.close();
        return encodeBytes(serialObj.toByteArray());
    }

    public static Object deserialize(String str) throws IOException {
        ObjectInputStream objStream = null;
        Object obj = null;
        if (str == null || str.length() == 0)
            return null;
        ByteArrayInputStream serialObj = new ByteArrayInputStream(decodeBytes(str));
        objStream = new ObjectInputStream(serialObj);
        try {
            obj = objStream.readObject();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return obj;
    }
}

public class ObjectSerializer {

    public static String serialize(Serializable obj) throws IOException {
        ByteArrayOutputStream serialObj = null;
        if (obj == null)
            return "";
        serialObj = new ByteArrayOutputStream();
        ObjectOutputStream objStream = new ObjectOutputStream(serialObj);
        objStream.writeObject(obj);
        objStream.close();
        return encodeBytes(serialObj.toByteArray());
    }

    public static Object deserialize(String str) throws IOException {
        ObjectInputStream objStream = null;
        Object obj = null;
        if (str == null || str.length() == 0)
            return null;
        ByteArrayInputStream serialObj = new ByteArrayInputStream(decodeBytes(str));
        objStream = new ObjectInputStream(serialObj);
        try {
            obj = objStream.readObject();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return obj;
    }
}
public static String encodeBytes(byte[] bytes) {
    StringBuffer strBuf = new StringBuffer();
    for (int i = 0; i < bytes.length; i++) {
        strBuf.append(((char) (((bytes[i] >>> 4) & 0xF) + ((int) 'a'))));
        strBuf.append((char) (((bytes[i]) & 0xF) + ((int) 'a')));
    }
    return strBuf.toString();
}

public static byte[] decodeBytes(String str) {
    byte[] bytes = new byte[str.length() / 2];
    for (int i = 0; i < str.length(); i += 2) {
        char c = str.charAt(i);
        bytes[i / 2] = (byte) ((c - 'a') << 4);
        c = str.charAt(i + 1);
        bytes[i / 2] += (c - 'a');
    }
    return bytes;
}
PendingAlarmReceiver.java

package com.sacstate.csc502.locationalert;

import com.sacstate.csc502.locationalert.ProximityAlertManager;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

/**
 * @author Sukanya. Broadcast receiver for time alarms.
 * *
 */
public class PendingAlarmReceiver extends BroadcastReceiver {

    LocationAlertActivity locationAlertActivity = new LocationAlertActivity();
    ProximityAlertManager proximityAlertManager;

    @Override
    public void onReceive(Context context, Intent intent) {
        proximityAlertManager = new ProximityAlertManager(context);
        long MILLISECS_IN_DAY = 86400000;
        int alertId = intent.getIntExtra(
                context.getString(R.string.request_code), -1);
        double latitude = intent.getDoubleExtra("lat", 91);
        double longitude = intent.getDoubleExtra("long", 181);
        String radius = intent.getStringExtra("radius");
        proximityAlertManager.setProximityAlert(latitude, longitude, radius,
                MILLISECS_IN_DAY, alertId);
    }
}
PendingAlertReceiver.java

package com.sacstate.csc502.locationalert;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.location.LocationManager;
import android.widget.Toast;

/**
 * @author Sukanya. Broadcast receiver for proximity alerts.
 *
 */
public class PendingAlertReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent arg1) {
        boolean enterLocation = arg1.getBooleanExtra(
                LocationManager.KEY_PROXIMITY_ENTERING,
                false);
        CharSequence text = "Entering";
        int duration = Toast.LENGTH_LONG;
        if (enterLocation) {
            Toast toast = Toast.makeText(context, text, duration);
            toast.show();
            NotifyUser(context);
        }
    }

    public void NotifyUser(Context context) {
        int icon = R.drawable.ic_launcher;
        String tickerText = "Notification";
        long when = System.currentTimeMillis();
        Notification notification = new Notification(icon, tickerText, when);
        notification.flags = Notification.FLAG_AUTO_CANCEL;
        notification.defaults = Notification.DEFAULT_VIBRATE |
                Notification.DEFAULT_SOUND;
        PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, null, 0);
    }
}
notification.setLatestEventInfo(context, "Location Alert",
    "Destination reached", pendingIntent);
NotificationManager notificationMgr = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationMgr.notify(1, notification);
package com.sacstate.csc502.locationalert;

import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.location.LocationManager;

public class ProximityAlertManager {
    LocationManager locationManager;
    String intentName = "com.sacstate.csc502.locationalert.LOCATION_ALERT";
    Context context;

    public ProximityAlertManager(Context context) {
        locationManager = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
        this.context = context;
    }

    public String checkForLocationProviders() {
        locationManager = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
        if (locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
            return locationManager.GPS_PROVIDER;
        else if (locationManager
            .isProviderEnabled(LocationManager.NETWORK_PROVIDER))
            return locationManager.NETWORK_PROVIDER;
        else
            return null;
    }

    public void setProximityAlert(double latitude, double longitude,
        String stringRadius, long expiration, int alertId) {
        Intent intent = new Intent(intentName);
        intent.putExtra(context.getString(R.string.request_code), alertId);
}
PendingIntent pendingIntent =
PendingIntent.getBroadcast(context.getApplicationContext(), alertId,
    intent, 0);
Float radius = Float.parseFloat(stringRadius);
locationManager.addProximityAlert(latitude, longitude, radius,
    expiration, pendingIntent);
}

public void removeProximityAlert(int alertId)
{
    Intent intent = new Intent(intentName);
    intent.putExtra(context.getString(R.string.request_code), alertId);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(context,
        alertId, intent, 0);
    locationManager.removeProximityAlert(pendingIntent);
}
import java.util.Calendar;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;

/**
 * @author Sukanya. Class responsible for setting time alarms that repeat every week.
 *
 */
public class RepeatingAlarmManager {
    AlarmManager alarmManager;
    String intentName = "com.sacstate.csc502.locationalert.RepeatingAlarmManager.ALARM_EVENT";
    Context context;

    public RepeatingAlarmManager(Context context) {
        alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
        this.context = context;
    }

    public void setDayBasedRepeatingAlarm(int day, int alarmId, double latitude, double longitude, String radius) {
        int numOfDaysToAdd = 0;
        Calendar calendar = Calendar.getInstance();
        int today = calendar.get(Calendar.DAY_OF_WEEK);
        if (day >= today)
            numOfDaysToAdd = day - today;
        else
            numOfDaysToAdd = (7 - today) + day;

        calendar.set(Calendar.DAY_OF_MONTH, calendar.get(Calendar.DAY_OF_MONTH) + numOfDaysToAdd);
        calendar.set(Calendar.HOUR_OF_DAY, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
Intent intent = new Intent(intentName);
intent.putExtra(context.getString(R.string.request_code), alarmId);
intent.putExtra("lat", latitude);
intent.putExtra("long", longitude);
intent.putExtra("radius", radius);
PendingIntent pendingIntent = PendingIntent.getBroadcast(
    context.getApplicationContext(), alarmId, intent, 0);
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
    calendar.getTimeInMillis(), 604800000, pendingIntent);
}

public void cancelRepeatingAlarm(int alarmId) {
    Intent intent = new Intent(intentName);
    intent.putExtra(context.getString(R.string.request_code), alarmId);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(
        context.getApplicationContext(), alarmId, intent, 0);
    alarmManager.cancel(pendingIntent);
}
SetAlertActivity.java

package com.sacstate.csc502.locationalert;

import java.util.List;
import android.preference.Preference;
import android.preference.PreferenceActivity;
import android.preference.Preference.OnPreferenceClickListener;
import android.provider.ContactsContract;
import android.util.SparseBooleanArray;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.database.Cursor;
import android.location.Address;
import android.location.Geocoder;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;

/**
 * @author Sukanya. Activity responsible for creating new alert entities and
deleting existing alert entities.
 */
public class SetAlertActivity extends PreferenceActivity {
    SharedPreferences defaultSharedPreferences;
    Intent callerIntent, calleeIntent;
    static final int DIALOG_ADDRESSERROR_ID = 0;
    static final int DIALOG_PROGRESS_ID = 1;
    static final int DIALOG_ADDRESSOPTIONS_ID = 2;
    static final int DIALOG_REPEATDAYS_ID = 3;
    static final int DIALOG_EDITADDRESS_ID = 4;
    static final int DIALOG_EMPTYADDRESS_ID = 5;
    static final int PICKCONTACT = 2;
    static final int FAVORITE_PLACE_LIST_ACTIVITY = 3;
    RepeatingAlarmManager repeatingAlarmManager;
    Preference addressPreference, repeatPreference;
    boolean[] repeatOnDays = new boolean[7];
boolean isRepeatTurnedOn = false;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    repeatingAlarmManager = new RepeatingAlarmManager(this);
    defaultSharedPreferences = SharedPreferencesProvider
defaultSharedPreferences(this);
    callerIntent = getIntent();

    setPreferences();
    addPreferencesFromResource(R.xml.preferences);
    setContentView(R.layout.preference_layout);

    final Button saveButton = (Button) findViewById(R.id.button_save);
    saveButton.setOnClickListener(new SetAlertActivitySaveButtonListener());

    final Button cancelButton = (Button) findViewById(R.id.button_cancel);
    cancelButton.setOnClickListener(new SetAlertActivityCancelButtonListener());

    addressPreference = (Preference) getPreferenceScreen().findPreference(
            "preference_address_key");
    addressPreference.setOnPreferenceClickListener(new AddressPreferenceListener());
    addressPreference.setSummary(defaultSharedPreferences.getString(
            getString(R.string.preference_address_key), ""));

    repeatPreference = (Preference) getPreferenceScreen().findPreference(
            "preference_repeat_days_key");
    repeatPreference.setOnPreferenceClickListener(new RepeatPreferenceListener());
}

private void setPreferences() {
    boolean isNewAlert = callerIntent.getBooleanExtra(
            getString(R.string.new_alert), true);
    SharedPreferences.Editor editor = defaultSharedPreferences.edit();

    if (isNewAlert) {
        editor.putBoolean(getString(R.string.preference_turnon_key), false);
        editor.putString(getString(R.string.preference_alertname_key), 
"");
        editor.putString(getString(R.string.preference_address_key), 
"");
        editor.putString(getString(R.string.preference_radius_key), 
"1000");
    }
editor.commit();
} else {
    int listItemPos = callerIntent.getIntExtra(
        getString(R.string.listitem_pos), -1);
    if (listItemPos != -1) {
        AlertEntity alertEntity = AlertListManager
            .getAlertEntityAtListPos(listItemPos);
        editor.putBoolean(getString(R.string.preference_turnon_key),
            alertEntity.enabled);
        editor.putString(getString(R.string.preference_alertname_key),
            alertEntity.name);
        editor.putString(getString(R.string.preference_address_key),
            alertEntity.address);
        editor.putString(getString(R.string.preference_radius_key),
            alertEntity.radius);
        editor.commit();
    }
}

protected Dialog onCreateDialog(int id) {
    Dialog dialog = null;
    switch (id) {
    case DIALOG_ADDRESSERROR_ID:
        AlertDialog.Builder addressErrorBuilder = new
            AlertDialog.Builder(this);
        addressErrorBuilder
            .setMessage("Unable to find address. Check
                network connection and provide valid address")
            .setNeutralButton("OK",
            new
                DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog,
                        int id) {
                        dialog.dismiss();
                    }
                });
        AlertDialog addressErrorDialog = addressErrorBuilder.create();
        return addressErrorDialog;
    case DIALOG_ADDRESSOPTIONS_ID:
        final CharSequence[] items = { "Use Contacts",}
"Use Favorite Places List", "Enter Address";
AlertDialog.Builder addressOptionsBuilder = new
AlertDialog.Builder(
    this);
addressOptionsBuilder.setTitle("Choose an option");
addressOptionsBuilder.setSingleChoiceItems(items, -1,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int item) {
            if (item == 0) {
                Intent intent = new
                Intent(Intent.ACTION_PICK,
                    ContactsContract.Contacts.CONTENT_URI);
                startActivityForResult(intent,
                    PICKCONTACT);
            } else if (item == 1) {
                Intent intent = new
                Intent(getBaseContext(),
                    FavoritePlacesListActivity.class);
                startActivityForResult(intent,
                    FAVORITE_PLACE_LIST_ACTIVITY);
            } else if (item == 2) {
                showDialog(DIALOG_EDITADDRESS_ID);
            }
        }
    });
AlertDialog addressOptionsDialog =
    addressOptionsBuilder.create();
return addressOptionsDialog;

case DIALOG_EDITADDRESS_ID:
    final EditText input = new EditText(this);
    AlertDialog.Builder editAddressBuilder = new
    AlertDialog.Builder(
        this);
editAddressBuilder
        .setTitle("Enter Address")
        .setView(input)
        .setPositiveButton("OK",
            new
            DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int whichButton) {
    String addressVal = (String) input.getText().toString();
    defaultSharedPreferences.edit().putString(getString(R.string.preference_address_key), addressVal).commit();
    if (!addressVal.isEmpty()) {
        dismissDialog(DIALOG_ADDRESSOPTIONS_ID);
    }
}).setNegativeButton("Cancel", null);
AlertDialog alertEditAddressDialog = editAddressBuilder.create();
return alertEditAddressDialog;

case DIALOG_REPEATDAYS_ID:
    boolean isNewAlert = callerIntent.getBooleanExtra(getString(R.string.new_alert), true);
    boolean[] checkedItems = null;
    if (!isNewAlert) {
        checkedItems = new boolean[7];
        int listItemPos = callerIntent.getIntExtra(getString(R.string.listitem_pos), -1);
        if (listItemPos != -1) {
            AlertEntity alertEntity = AlertListManager.getAlertEntityAtListPos(listItemPos);
            for (int i = 0; i < 7; i++)
                checkedItems[i] = alertEntity.repeatOnDays[i];
        }
    }
    final CharSequence[] days = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" };
    AlertDialog.Builder repeatDaysDialogBuilder = new AlertDialog.Builder(this);
    repeatDaysDialogBuilder.setTitle("Repeat");
repeatDaysDialogBuilder.setMultiChoiceItems(days, checkedItems, new DialogInterface.OnMultiChoiceClickListener() {
    public void onClick(DialogInterface arg0, int arg1, boolean arg2) {

    }
});
repeatDaysDialogBuilder.setPositiveButton("OK", new RepeatDialogOkButtonListener());
repeatDaysDialogBuilder.setNegativeButton("Cancel", null);
AlertDialog repeatDaysDialog = repeatDaysDialogBuilder.create();
return repeatDaysDialog;

case DIALOG_EMPTYADDRESS_ID:
    AlertDialog.Builder emptyAddressBuilder = new AlertDialog.Builder(this);
    emptyAddressBuilder.setMessage("Address field of the chosen contact is empty")
        .setNeutralButton("OK", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                dialog.dismiss();
            }
        });
    AlertDialog alertDialog = emptyAddressBuilder.create();
    return alertDialog;
    default:
    dialog = null;
    } return dialog;

@override
public void onActivityResult(int reqCode, int resultCode, Intent data) {
    super.onActivityResult(reqCode, resultCode, data);
switch (reqCode) {
    case (PICKCONTACT):
        if (resultCode == RESULT_OK && data != null) {
            dismissDialog(DIALOG_ADDRESSOPTIONS_ID);
            Uri contactPicked = data.getData();
            Cursor c = managedQuery(contactPicked, null, null, null, null);
            if (c.moveToFirst()) {
                Cursor address = getContentResolver()
                    .query(ContactsContract.CommonDataKinds.StructuredPostal.CONTENT_URI,
                            ContactsContract.CommonDataKinds.StructuredPostal.CONTACT_ID
                    + " = "
                    + c.getString(c
                        .getColumnIndex(ContactsContract.Contacts._ID)),
                            null, null);
                String addressVal = "";
                String temp;
                if (address.moveToFirst()) {
                    if ((temp = address
                        .getString(address
                            .getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.POBOX
)))) != null)
                        addressVal += temp + ", ";
                    if ((temp = address
                        .getString(address
                            .getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.STREET
)))) != null)
                        addressVal += temp + ", ";
                    if ((temp = address
                        .getString(address
                            .getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.CITY))
)) != null)
                        addressVal += temp + ", ";
                    if ((temp = address
                        .getString(address
                            .getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.STATE
)))) != null)
                        addressVal += temp + ", ";
                    if ((temp = address
                        .getString(address
                            .getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.COUNTRY
)))) != null)
                        addressVal += temp + ", ";
                }
            }
            if (addressVal.trim().length() == 0) {
                addressVal = "";
            }
            return addressVal;
        }
    return null;
}
addressVal += temp + ", ";
if (temp = address.getString(address
.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.POSTC
TRY))) != null)
    addressVal += temp + ", ";
    if (temp = address.getString(address
.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.COUN
TRY))) != null)
    addressVal += temp;
    if (addressVal.trim().endsWith(",
))
        addressVal = addressVal.substring(0,
            addressVal.length() - 2);
        defaultSharedPreferences.edit().putString(
            getString(R.string.preference_address_key),
            addressVal).commit();
addressPreference.setSummary(addressVal);
} else
    showDialog(DIALOG_EMPTYADDRESS_ID);
} else
    break;
    case FAVORITE_PLACE_LIST_ACTIVITY:
        if (resultCode == RESULT_OK && data != null) {
            dismissDialog(DIALOG_ADDRESSOPTIONS_ID);
            String addressVal = data
                .getStringExtra(getString(R.string.favorite_place));
            defaultSharedPreferences
                .edit()
                .putString(getString(R.string.preference_address_key),
                    addressVal).commit();
            addressPreference.setSummary(addressVal);
        }
    }
}

class RepeatDialogOkButtonListener implements
    DialogInterface.OnClickListener {
    public void onClick(DialogInterface dialog, int which) {
        int days_in_week = 7;
        SparseBooleanArray checkedDays = ((AlertDialog) dialog)
            .getListView().getCheckedItemPositions();
        for (int i = 0; i < days_in_week; i++) {
            if (checkedDays.get(i)) {

isRepeatTurnedOn = true;
repeatOnDays[i] = true;
}
}
}
}
}

class AddressPreferenceListener implements OnPreferenceClickListener {
    public boolean onPreferenceClick(Preference arg0) {
        showDialog(DIALOG_ADDRESSOPTIONS_ID);
        return true;
    }
}

class RepeatPreferenceListener implements OnPreferenceClickListener {
    public boolean onPreferenceClick(Preference arg0) {
        showDialog(DIALOG_REPEATDAYS_ID);
        return true;
    }
}

class SetAlertActivitySaveButtonListener implements View.OnClickListener {
    public void onClick(View arg0) {
        calleeIntent = new Intent();
        calleeIntent.putExtra(getString(R.string.new_alert), callerIntent.getBooleanExtra(getString(R.string.new_alert), true));
        calleeIntent.putExtra(getString(R.string.listitem_pos), callerIntent.getIntExtra(getString(R.string.listitem_pos), -1));
        calleeIntent.putExtra(getString(R.string.repeat_on_days), repeatOnDays);
        calleeIntent.putExtra(getString(R.string.is_repeat_turnedon), isRepeatTurnedOn);
        new ForwardGeoCoder().execute(defaultSharedPreferences.getString(getString(R.string.preference_address_key), ""));
    }
}

class SetAlertActivityCancelButtonListener implements View.OnClickListener {
    public void onClick(View arg0) {
        removeDialog(DIALOG_REPEATDAYS_ID);
        setResult(RESULT_CANCELED);
    }
}
private class ForwardGeoCoder extends AsyncTask<String, Void, Address> {

    Geocoder fwdGeoCoder = new Geocoder(getApplicationContext());

    @Override
    protected Address doInBackground(String... address) {
        List<Address> locations = null;

        if (Geocoder.isPresent()) {
            try {
                locations =
                    fwdGeoCoder.getFromLocationName(address[0], 1);
            } catch (Exception e) {
                locations = null;
            }
        }

        if (locations != null && locations.size() > 0)
            return locations.get(0);
        else
            return null;
    }

    @Override
    protected void onPostExecute(Address address) {
        super.onPostExecute(address);
        if (address != null) {
            calleeIntent.putExtra(getString(R.string.latitude),
                    address.getLatitude());
            calleeIntent.putExtra(getString(R.string.longitude),
                    address.getLongitude());
            setResult(RESULT_OK, calleeIntent);
            finish();
        } else
            showDialog(DIALOG_ADDRESSERROR_ID);
    }
}
SetFavoritePlaceActivity.java

package com.sacstate.csc502.locationalert;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceActivity;
import android.view.View;
import android.widget.Button;

/**
 * @author Sukanya. Activity responsible for creating new favorite place
 *         entities and editing existing favorite place entities.
 *
 */
public class SetFavoritePlaceActivity extends PreferenceActivity {
    SharedPreferences defaultsharedPref;
    Intent callerIntent, calleeIntent;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        defaultsharedPref = SharedPreferencesProvider
            .getDefaultSharedPreferences(this);
        callerIntent = getIntent();
        setPreferences();
        addPreferencesFromResource(R.xml.favoriteplace_preferences);
        setContentView(R.layout.preference_layout);
        final Button saveButton = (Button) findViewById(R.id.button_save);
        saveButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                calleeIntent = new Intent();
                calleeIntent.putExtra(
                    getString(R.string.new_favoriteplace_item),
                    callerIntent.getBooleanExtra(
                        getString(R.string.new_favoriteplace_item),
                        true));
                calleeIntent.putExtra(getString(R.string.listitem_pos),
                    callerIntent.getIntExtra(
                        getString(R.string.listitem_pos), -1));
                setResult(RESULT_OK, calleeIntent);
                finish();
            }
        });
    }
}
final Button cancelButton = (Button) findViewById(R.id.button_cancel);
cancelButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        setResult(RESULT_CANCELED);
        finish();
    }
});

private void setPreferences() {
    boolean isNewPlace = callerIntent.getBooleanExtra("
        getString(R.string.new_favoriteplace_item), true);
    SharedPreferences.Editor editor = defaultSharedPref.edit();

    if (isNewPlace) {
        editor.putString("
            getString(R.string.preference_favoriteplace_name_key), "
            editor.putString("
                getString(R.string.preference_favoriteplace_address_key),""
            editor.commit();
    } else {
        int listItemPos = callerIntent.getIntExtra("
            getString(R.string.listitem_pos), -1);
        if (listItemPos != -1) {
            FavoritePlaceEntity favoritePlaceEntity = FavoritePlacesListManager
                .getFavoritePlaceEntityAtListPos(listItemPos);
            editor.putString("
                getString(R.string.preference_favoriteplace_name_key),
                favoritePlaceEntity.name);
            editor.putString("
                getString(R.string.preference_favoriteplace_address_key),
                favoritePlaceEntity.address);
            editor.commit();
        }
    }
}
SharedPreferencesProvider.java

package com.sacstate.csc502.locationalert;

import android.content.Context;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;

/**
 * @author Sukanya. Class responsible for retrieving default shared preferences file of the application.
 */
public class SharedPreferencesProvider {

    static SharedPreferences defaultSharedPrefs;

    private SharedPreferencesProvider() {
    }

    public static SharedPreferences getdefaultSharedPreferences(Context context) {
        if (defaultSharedPrefs == null) {
            defaultSharedPrefs = PreferenceManager.getDefaultSharedPreferences(context .getApplicationContext());
            return defaultSharedPrefs;
        }
    }
}
BIBLIOGRAPHY

[1] Reto Meier, “Professional Android 2 application development”


[4] Conder, Shane, Android wireless application development, 2nd ed


[8] Geo-Reminders iPhone application

Available :
https://github.com/apache/pig/blob/89c2e8e76c68d0d0abe6a36b4e08dcd56979796f/src/org/apache/pig/impl/util/ObjectSerializer.java