Group 10 Android Maintenance Manual

February 12, 2014

SE.10.MAN_ANDROID.1

Version: 1.0
Status: Release

Contributor Name	**Role**
Daniel Clark	Project Lead
Mark Lewis	QA Manager & Web Developer
Charles Newey	Deputy Project Lead & Android Developer
Martin Ferris	Android Developer
Ashley Iles	Android Developer
Kenny Packer	Android Developer
Stephen McFarlane	Deputy QA & Web Developer
Kieran Palmer	Web Developer

Department of Computer Science,
Llandinam Building,
Aberystwyth University,
Aberystwyth,
Ceredigion,
SY23 3DB

Contents

1 INTRODUCTION

1.1 Purpose of This Document

To aid with future maintenance of the Android application, and to document the idiosyncrasies and intricacies of the Java code used to develop it.

1.2 Scope

This document should be read by any party that is involved in future maintenance of the application. This document contains details of a number of important factors, including:

- Coding style
- Supported Android SDK versions
- Variable naming schemes
- Main data structures
- Suggestions for improvements

1.3 Objectives

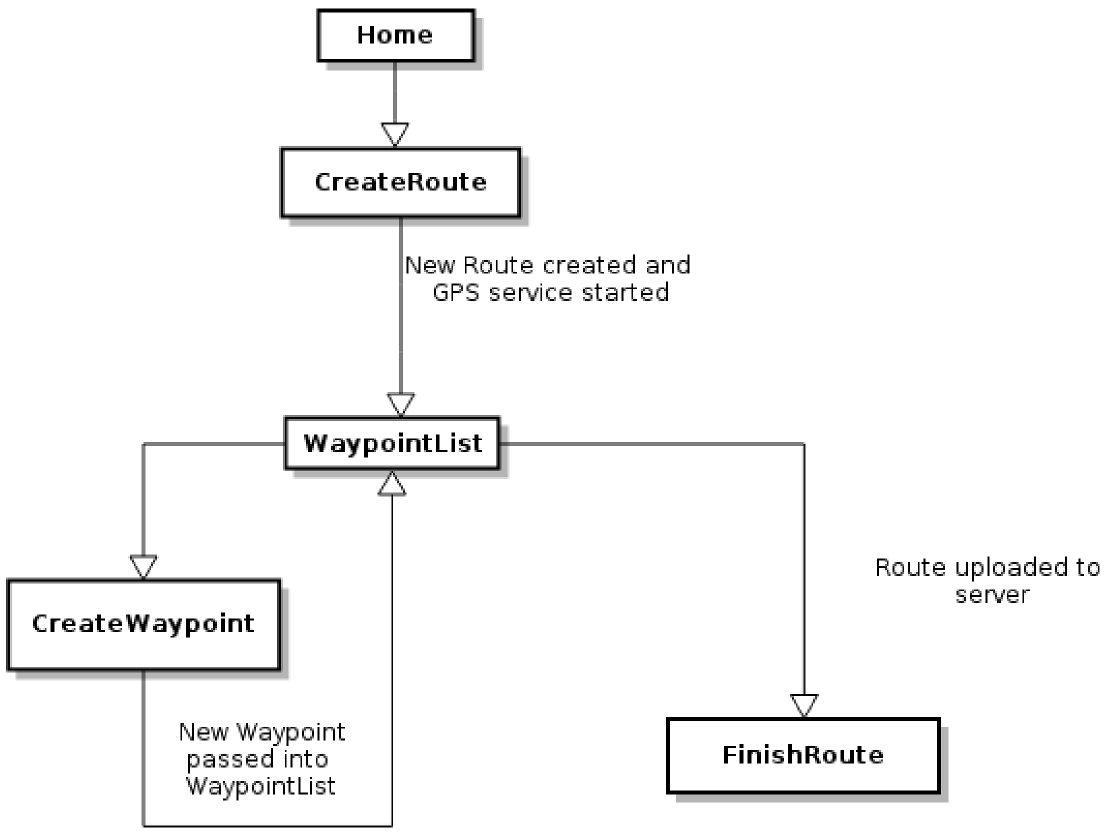
- To give an overview into the structure of the application
- To aid the package maintainer in spotting problem areas in the project's code base
- To provide the package maintainer a reference for complex areas in the code base

2 Program information

2.1 Program Description

The Stumblr Android application is written in Java and is essentially built to help with the creation and upload of walking tours. It allows the user to record a "breadcrumb trail" of GPS coordinates, and gives the ability to add "Points of Interest" (Waypoints) along the way. The user can attach metadata to each Waypoint - including a title, a description, a photo (from the user's gallery or camera), and a GPS location. Before uploading, other metadata is attached to the walk - including time taken and cumulative distance. Once the walk is complete, it is then uploaded (in JSON format) to a remote server, where the data is processed and displayed on a web interface.

2.2 Data flow diagram


Home
CreateRoute
New Route created and GPS service started
WaypointList
CreateWaypoint
New Waypoint passed into WaypointList
Route uploaded to server
FinishRoute

3 Data Classes

This section details the significant or complex methods and data structures in each class - large parts of the code base are very well-commented and self-explanatory, so it is only the more complex areas that are mentioned in this document.

3.1 StumblrData

3.1.1 Significant methods

- **public string sanitiseStringInput**
  This method takes a String object and replaces all forbidden characters (more specifically, characters that do not belong to the set "a-z", "A-Z", "0-9", ",.!?:;" and "-". This ensures that String data is safe to transmit to the web server.

3.2 Route

The Route class is the main container for the metadata of the walk - including the description, timestamps, the list of coordinates for the breadcrumb trail, and the list of Waypoints (a datatype that will be covered further later in this document).

3.2.1 Significant data structures

- **private Stack<Location> coordinates;**
  This holds the "breadcrumb trail" - a Stack of Location objects (from the Android API). This is where the Location objects are stored when they are obtained from the GPSService class. Location objects are passed from the GPSService class by using Broadcastsm and then filtering by intent. (This is covered in detail later).
- **private LinkedList<Waypoint> route;**
  This holds the list of Waypoint objects - including an instance of a Bitmap image (from the Android API), and Strings relating to the title and description of the Waypoint.

3.2.2 Significant methods

- **public float getDistance**
  This method calculates the cumulative distance between all of the "breadcrumbs" in the 'coordinates' Stack. The method loops through the list and calls the Android API method **Location.distanceBetween()** on each successive pair of coordinates,. Returns a **float** containing the total distance between each pair of coordinates.
- **public void writeToParcel**
  Writes the current contents of the Route object to a **Parcel** object (Android API), to allow the Route to be passed between Activities. Required for implementing the **Parcelable** interface.
- **public void readFromParcel**
  Reads a Parcel into the current Route object. Required for implementing the **Parcelable** interface.

3.3 Waypoint

3.3.1 Significant methods

- **public void writeToParcel**
  Writes the current contents of the Waypoint object to a **Parcel** object (Android API), to allow the Route to be passed between Activities. Required for implementing the **Parcelable** interface.

- **public void readFromParcel**
  Reads a Parcel into the current Waypoint object. Required for implementing the **Parcelable** interface.

4 Activity Classes

4.1 AbstractActivity

This class contains all of the common code elements that are used in the other Activities. This includes the code that display the "cancel confirmation" alert dialogue. This class is largely self-explanatory.

4.2 CreateRoute

4.2.1 Significant methods

- **public void startWaypointListIntent**
  This starts the WaypointList Activity after checking that the input fields of the Activity are the correct length.

4.3 CreateWaypoint

This Activity is called from the WaypointList activity - and this is where the user can create Waypoints to correspond with Points of Interest (POIs) during the walk.

4.3.1 Significant methods

- **public void stumblrOnCreate**
  This method is essentially the Activity's constructur - it checks the Bundle data passed into the Activity - checking for a Location object or a Waypoint object. The object is then loaded into the appropriate instance variables (depending on the type). The last thing this method does is use the Android API to check if certain capabilities are enabled; namely, a camera and gallery.

- **public void finishWaypoint**
  This just validates the data entered on-screen, warning the user with a **Toast** if any data entered is invalid. A timestamp is then applied to the current Waypoint, and the Waypoint object is put into a Bundle as a return value for the Activity - this Bundle is returned to WaypointList (as it is instantiated with **startActivityForResult()**).

- **public void getImage**
  Gets an image from user's camera or gallery, after checking which is available. An Activity is dispatched (based upon what is available) using **startActivityForIntent**.

- **protected void onActivityResult**
  This method handles return values from the camera or gallery Activities dispatched in the previous method. Depending on the activity that returned the image, it will be resized and cast to a **Bitmap** for easy conversion to Base64, and submission (via JSON) to the web server.

4.4 FinishRoute

This Activity is responsible for converting the (completed) Route object to JSON and uploading it to the web server, ready for displaying on the web interface. As the data structures dealt with in this class are generally

4.4.1 Significant methods

- **public void stumblrOnCreate**
  This method is fairly simple - it unbundles the Route object from the extras passed in from the calling Intent, and puts it into memory. The remaining Route metadata is then calculated and displayed onscreen.
- **public void postData**
  This method simply uses the Android API to check if the device has an internet connection - if so, the NetworkTast is executed, and the data is sent.
- **public JSONObject getData**
  This converts the Route object into JSON ready for transmission to the web server. This method uses the **JSONObject (org.json.JSONObject)** class from the standard Java JSON libraries.
- **public String encodeToBase64**
  This is fairly simple - a **ByteArrayOutputStream** decodes a **Bitmap** object into raw data - which is then encoded using Base64

4.5 GPSService

GPSService uses a Foreground Notification (to avoid getting killed by Android) and implements **Location-Listener** so that it is able to subscribe to Location update (provided by GPS). The **Location** objects are then sent back to the main Activity by using a Broadcast.

4.5.1 Significant methods

- **public int onStartCommand**
  This is a rather straightforward method, but it looks a little complicated. It subscribes the current class to receive **Location** updates and places a persistent **Notification** in the user's Notification Centre. **Be aware: There is officially deprecated code inside this method so that the persistent Notification can be shown on older versions of Android.**
- **public void onDestroy**
  This method tells the class to stop receiving Location updates when killed - this ensures that the device's battery is not wasted.
- **public void onLocationChanged**
  This method is very simple, but its function may not be immediately obvious. It is called when a **Location** update is received by the **GPSService** class, and it bundles the received **Location** object and sends it as a **Broadcast** to the system, where the message is picked up by **WaypointList** and added to the current Waypoint.

4.6 Home

The Home class contains no complex methods as it is approximately 35 lines in length and only contains code for one button!

4.7 WaypointList

This is the main Activity of the application in that most of the user's time is spent using it. From this Activity, the user can view, add to, and edit the current list of Waypoints. The user can also decide to complete the walk, dispatching the FinishRoute Activity.

4.7.1 Significant data structures

- **private ListView listView**
  This ListView object is used to display and manipulate the Route's Waypoints on-screen. The user can view a list of the current Waypoints, or edit the

4.7.2 Significant methods

- **protected void onActivityResult**
  This method is called when a dispatched Activity (in this case, CreateWaypoint) exits and returns with a valid Waypoint object - the new Waypoint is added to the list and the ListView is refreshed.

- **private void startGPSService**
  There is a private class declaration within this method that contains methods to add coordinates received from the GPSService to the current trail of **Locations**. If GPS is disabled, the method prompts the user to enable it.

- **public void onDestroy**
  Called before the Activity is destroyed (part of the Android application lifecycle). This sends a signal to the running GPS service to halt and exit (thus removing the **Notification** at the top of the user's screen).

5 Limitations of the Application

There are a number of limitations to the application in question, the principal of which are the Android API versions that it supports. Currently, the application supports (and has been fully tested on) API/SDK versions **8** to **22** - which correlates to Android versions **2.2** through **4.4**.

There are also a number of important considerations when it comes to battery and memory usage. Each Waypoint (including thumbnail image) is stored entirely in working memory (RAM) - theoretically, this limits the number of Waypoints that a low-RAM device can hold; although in practice, the amount of data in memory (even on large walks) is relatively small, and memory usage is a very minor consideration. Certainly, we experienced no issues on an older Android **2.3** device with severe memory limitations, with walks of 20 Waypoints and above.

The application is also rather battery intensive on longer walks; the GPS service, while being useful for regularly fetching location data - uses an enormous amount of battery power, particularly if the GPS signal is bad (due to overcast weather, large buildings in the surrounding area, etc). This is sufficient to flatten some more power-hungry devices with 5-6 hours' usage (tested overnight on a tablet device).

6 Interaction with Remote Server

The interaction with the server is kept as simple as possible. The Java objects created within the application are converted to JSON (a human-readable format useful when transmitting complex objects), and transferred to the server via HTTP POST. The HTTP interaction is completed with the Apache Commons HTTP libraries, and the code used to interact with the server is currently in the FinishRoute class.

7 Building and Testing

This section details the tools and methods that were used to create and compile the application and documentation.

7.1 Building

The build tools used in the creation of the Java code base were Android Studio, coupled with the Gradle build system. The supplementary files inside the Git repository are mostly to ensure proper functioning of Gradle and the Android Studio IDE.

7.2 Testing

The testing code for this assignment was written using JUnit - a testing suite that complements the Java language.

7.3 Documentation

All documentation for this project (where possible) has been written and rendered using LaTeX. LaTeXcan be a challenge to write for new users, so a procedure was defined for team members who struggle getting it to work; writing documents in MarkDown and then converting them. MarkDown is a minimal, intuitive markup language (also used on GitHub for formatting text files) that aims to be easy to write. This was excellent for our needs, as MarkDown syntax can be easily converted to LaTeXsyntax using a system like PanDoc.

Markdown: `https://daringfireball.net/projects/markdown/`
PanDoc: `http://johnmacfarlane.net/pandoc/`

7.4 References to external sources

References to external sources are usually contained inside inline comments in the Java code; *for example*:

```
// See:  somewebsite.com/url/resource.html
```

8 REFERENCES

[1] Pavlic, T. Programming Assignment template for LaTeX.
http://www.latextemplates.com/template/programming-coding-assignment.
Accessed: Oct 23, 2013.

9 VERSION HISTORY

Author	**Date**	**Version**	**Change made**
CCN	07/11/2013	1.0	Updated order of authors on cover