Feeling Glassy:
An Analysis of Developing for Google Glass

Dakota Gallimore
Department of Mathematics and Computer Science
University of Virginia, College at Wise
Wise, VA USA

Joseph LaCava
Department of Mathematics and Computer Science
University of Virginia, College at Wise
Wise, VA USA

Dr. Abrar Qureshi
Department of Mathematics and Computer Science
University of Virginia, College at Wise
Wise, VA USA

Abstract - *Google Glass is considered a front-runner of mobile computing. It functions as a wearable device that, as the name suggests resembles a pair of glasses, but the abilities of the device prove that this is no ordinary pair of spectacles. Understanding that this device stands to be the new and, currently, unrivaled device for the future of mobile computing, those with the ability to program for it stand to enter into a market that has limitless potential. The proposed paper serves as an analysis of the development process and attempts to outline the pieces that are required for both Android and Glass.*

Keywords: Activities, Services, Google Glass, Intents, Mirror API, Google Development Kit (GDK).

1. Introduction

Programming for Google Glass starts as any other programming project would, with the SDK. Google is currently in the stages of developing their own Android Studio, but for the purposes of coding for Glass we, the developers, utilized the existing Eclipse[1] and the ADT plugin bundle. Both, the Android SDK and the Android Studio, are available to download for free on the Google development website. As one might suspect from the IDE, Java[2] is the suggested language used in Android programming. However, for those who prefer C/C++ or simply wish to reuse existing libraries, Google also offers a NDK, but does not suggest its use to program in C/C++ and states using native code does not typically increase application performance. All of these and more is available of Google's developer site. The sample project that is later referenced in this paper was built in the Android SDK environment, the Eclipse IDE and Android ADT plugin bundle. Further detail of how to set up the IDE and importing the appropriate libraries will be covered later in this paper.

2. Project requirements

Assuming that the steps to download the SDK above were followed correctly and everything is working, if issues please consult the support provided by Google's website, it is time to download the appropriate library for GDK. Under Window, in the menu bar, select the Android SDK Manager which will pull up a window similar to the one below.

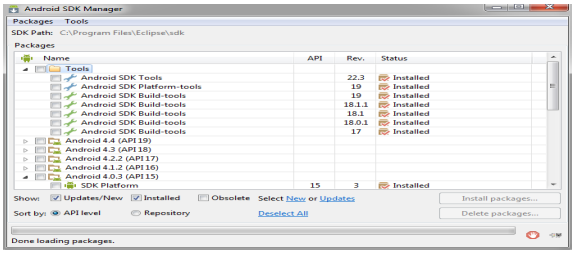

Figure 1. Android SDK Manager.

Make sure the most recent SDK tools have been downloaded and installed. All of the programming for Glass is through the Android 4.0.3 or API15, as is depicted in the screenshot below. You will need to download the entire API, excluding the samples at your prerogative, so it may be a better decision to do all of them at once. It should be noted that in the extras folder resides all of the legal documentation for android development. It may be a good idea to download and install these as well.

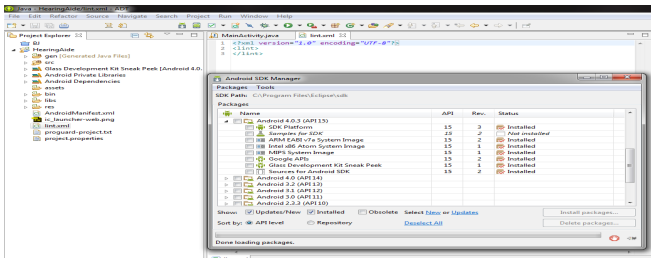

Figure 2. Android API 15

The Figure 2 shows the Android API 15 open and as you can see installed. If there any problems installing or

locating the correct API bundles please contact Google, or simply refer to their developer website. To continue into the depth of the GDK coding process, we have to give a few descriptions of the parts found and implemented in Android development.

3. Development parts[4]

3.1 Activities

Activities are the main building blocks of any android project. Activities handle anything having to do with user interaction and maintaining the user interface thread. Activities follow a lifecycle process. Developers can use this to contain actions performed within their projects. Such actions could include shutting down a running service for power management or memory optimization and management.

3.2 Services

Services handle background tasks much like AsyncTasks, however it is important to note that AsyncTasks are more suited for short term processes like downloading a file and then updating the UI thread to reflect completion whereas services are fitted for long running background tasks. This could be anything from audio playback to polling a WiFi connection in a set interval.

3.3 Fragments

Fragments are also play a keen role in android development and while Glass is based off android 4.0.3, they do not have a substantial role in Glass development but are worth noting. Fragments are a part of an activity and you can have multiple nested fragments per activity to give multiple viewpoints within an activities UI thread. This means all fragments share the same lifecycle state as their parent activity. Fragments were introduced in android 3.0 as a mean to extend the functionality and dynamic (*dynamic-like*) nature of android activities.

4. Glass environment[5]

The Glass environment is based off a timeline like implementation where events in the past are generated and stored on the left side of the home screen and present or upcoming events are stored. This is where Glass steps away from its fundamental android roots and becomes one in its own. Developers can publish and delete items pertaining to their application onto the Glass timeline. This being said the Glass development standards still apply regardless if the user is interacting with an activity or a timeline card. A time card is an instance of a developer's application being placed on the main timeline. Time cards allow applications to handle many tasks in the background like audio processing.

4.1 Menu Items

Menu items allow users to request actions related to the timeline cards. They are one of the most interesting services included in Glass and allow the user to interact directly with any timecard that is currently displayed. Android comes with many built-in functionalities, such as sharing images, replying to message, all of which may be command through either the touch panel or said aloud. The GDK allows developers to create their own menu items, thus increasing customization and fluidity inside the software.

4.2 Mirror API[5]

The Mirror API is a tool Google has created to allow developers to create services for Google Glass without having the software directly on the device. Google affectionately labels the modified software as "Glassware." The Mirror API allows everything to function like a web based service, increasing productivity and versatility, it does all of this without actually running code on the Glass itself. The Mirror API serves as a means to simply display the data without too much having to cross through Google Glass' connection to the internet. It utilizes a REST API method which essentially means that instead of using arbitrary verbs to pull data it instead utilizes a simpler uniform selection of the HTTP methods 'GET', 'POST', 'PUT', and 'DELETE,' but further detail into the precise command interfaces may be found on Google's developers website. Since the project referenced in this paper was not created using the Mirror API, it would be interesting to note, the developer website also includes sample projects in the following languages: "Go[3]," "Java[2]," ".NET[6]," "PHP[7]," "Python[8]," and "Ruby[9]."

The functionality of the Mirror API differs only slightly for each aspect of the Glass interface. Since the Mirror API does not have hardware access its functionality is limited in regards to the Glass Development Kit. However, a Glass user will be able to push content to Glassware that is using the Mirror API method, regardless of hardware access. A Glass user simply has to tie their Google id to Glassware that uses the Mirror API. The Glassware will then start to push content to Google's servers which will filter content to the appropriate end user. In addition, the Mirror API handles any displaying of data necessary in any glassware action. As was previously stated, the following project does not implement the Mirror API.

5. Project

The project encompasses the steps from creating a Glass application, "Glassware," to the finalization of the publishing of the finished project. As well as outlining the essential pieces implemented for functionality. The goal of this project is to improve the everyday life of the hearing impaired. We developed this with the hopes of evolving

into an all-around accessibility application. For now, the application caters to the hearing impaired. This project will act as the middleman between the speaker and the user. Much like movies have captions for the hearing impaired, our application will provide captions to the user for people around them.

When assessing which development tools to use, we knew using the GDK would be the most reasonable. This will allow our application to be used and accessed by the user even if no phone is connected or no data connection is available by using the built in voice recognition software. Due to the constraints of the deployment environment. We wanted to design the application to run only when the user wanted it. We opted for an activity based approach over a service based approach. This requires the user to launch the activity every time they need it but saves battery and allows the user to control what actually gets displayed. The user will queue up the application via the Glass voice menu. Since every application within the voice menu must contain a voice trigger we wanted to make ours easy and simple to pronounce.

```
<intent-filter >
    <action android:name="com.google.android.glass.action.VOICE_TRIGGER" />
</intent-filter>
<meta-data
    android:name="com.google.android.glass.VoiceTrigger"
    android:resource="@xml/my_voice_trigger" />
```

Figure 3. XML queuing voice trigger.

The voice trigger used by the Glass operating system is declared in the main AndroidManifest.xml file on a per activity base. The meta data contains the string information to be prompted to the user. The meta data tag points to our xml file located in our applications resource folder titled xml.

```
<?xml version="1.0" encoding="utf-8"?>
<trigger keyword="@string/glass_voice_trigger">

    <!-- Contains contraints to launching Glassware -->
    <constraints />

    <trigger keyword="@string/glass_voice_trigger">
        <input prompt="@string/glass_voice_prompt" />
    </trigger>

</trigger>
```

Figure 4. Voice trigger content

my_voice_trigger.xml contains the actual voice trigger content. Since we have no constraints determining if we can launch our application we simply point to our string resources that we give to the voice recognition activity. The Glass trigger keyword is aptly set as “caption” and the prompt simply tells the user that the Glass is listening and ready to receive voice input. As of the Glass operating system XE16+, when they moved to the Android operating system KitKat, applications must use one of the approved Glass command words listed at https://developers.google.com/glass/develop/gdk/reference/com/google/android/glass/app/VoiceTriggers.Command or add a permission named development in the applications manifest file (allows any command keyword to launch your app) to allow application launches on Glass.

```
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.google.android.glass.permission.DEVELOPMENT" />
```

Firgure 5. Voice trigger development permission

5.1 Touch Interface

Glass also allows for touch interaction of the voice menu items. This allows the users to interact with our application without needing to use voice. The user will use our intuitive gesture system where they can take two fingers and swipe towards the Glass screen to queue up a new caption. The user will need to do this for each new caption they want to see. This allows for user control when they want it and for the application to conserve power when the user is not using it due to not having a background service running and not allowing the processor to go into deep sleep. The user interface will fit into the Glass design restrictions by being quick and easy to read.

5.2 Voice Recognition

We call the voice recognition activity in the onResume() method of our application since the onCreate() method will call the onResume() method upon completion, this insures we gather voice data when the activity is resumed and created. We overwrite the main view screen each time the voice recognition activity is called if the results are not null.

```
    @Override
    protected void onResume()
    {
        super.onResume();
        if(voiceResults.get(0) != null)
        {
            voiceResults = getIntent().getExtras().getStringArrayList(RecognizerIntent.EXTRA_RESULTS);
            textForViewing.setText(voiceResults.get(0).toString());
        }
        else
            textForViewing.setText("Voice not captured");
    }
```

Figure 6. Calling Voice Recognition .

The user can queue up the Glass voice recognition activity manually from a menu item. This allows them to add captions when they want them and end a caption when they don't want them. We do this by passing an Intent to the voice recognition activity and waiting for a result from the VR activity.

```
47  private static final int SPEECH_REQUEST = 0;
48  private void launchSpeechRecognizer()
49  {
50      Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
51      .putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
52      startActivityForResult(intent, SPEECH_REQUEST);
53  }
```

Figure 7. Passing the Intent

launchSpeechRecognizer() is the method called by the menu listener that simply creates a voice recognition intent setting the language model to free form and starting the voice recognition activity for results.

```
54  @Override
55  protected void onActivityResult(int requestCode, int resultCode,
56          Intent data) {
57      if (requestCode == SPEECH_REQUEST && resultCode == RESULT_OK) {
58          results = data.getStringArrayListExtra(
59                  RecognizerIntent.EXTRA_RESULTS);
60          //String spokenText = results.get(0);
61          //voiceResults.get(0) = spokenText;
62          if(results.get(0) != null)
63              voiceResults.set(0, results.get(0));
64      }
65      super.onActivityResult(requestCode, resultCode, data);
66  }
```

Figure 8. Getting results.

Upon getting the results from the activity it automatically will recall onResume() which will update the UI thread from the zero index of voiceResults string array, so upon results from the voice recognizer we set the zero index of the voiceResults to the new value received from the Glass VR. This is the application and how we handle the many interactions from the user.

6. Product release

As many may have recognized the stages are summarized as so: Stage 1, Stage 2, Profit. Luckily, this isn't so far off from the truth! The methods implemented by Google to publish and profit from your application is incredibly simple. All Glassware and, possibly, applications developed with GDK, will be distributed through Google's MyGlass website. As of this publication, Google has not yet released how they plan to receive GDK developed apps. However, if the Glassware is developed with the Mirror API, the Google develop website has a link to an online application form. Google is requiring Glassware to be submitted for review before they are published on the MyGlass website. Those who are developing with the Mirror API may wish to reference checklist that Google has posted online, that simply outlines the requirements and practices that are to be implemented before the review process.

Google has forbidden any commercial profit from Glassware and Glass applications, but upon reading the legal documentation on the MyGlass website, we expect this to change in either the near future or upon the general public release. The MyGlass has legal documentation prepared for Google Wallet, and if it is set up anything like the existing Google Play Market, then a developer will need to create a Google Wallet account and the funds will transfer directly into the account.

7. Conclusion

With the ease of programming and the predefined API interfaces, not to mention the sheer number of APIs provided, Google Glass market stands to be as massive and successful as the Android/Google Play is currently. The methods of programming for Glass, as noted above, serve as a breach into the new emerging mobile market, giving developers the means to potentially make a name for themselves, which could also prove to be quite lucrative. Google continues to innovate with newer pursuits in mobile computing, opening the world for the general user, allowing them access to tools that are virtually limitless. It is impossible to reject such innovations based on their own merit. Glass proves these statements true and leads the frontier in mobile computing, making the call for quality application that much more emphatic and necessary, truly, this is the future and for those reading, now is the time to break out from the wires of tradition and move into the new horizon of mobile computing.

8. References

[1] The Eclipse Foundation. (n.d.) Retrieved December 9, 2013 from the Eclipse website: http://www.eclipse.org/

[2] Oracle Inc. (n.d.) Retrieved December 9, 2013 from the Java website: http://www.java.com/en/

[3] The Go Authors. (n.d.) Retrieved December 9, 2013 from the Go website: http://golang.org/

[4] Google Inc. (n.d.) Retrieved December 9, 2013 from the Google Developer website: https://developers.google.com/

[5] Google Inc. (2013, November 22) Retrieved December 9, 2013 from the Glass Developer website: https://developers.google.com/glass/

[6] Microsoft Inc. (2008) Retrieved December 9, 2013 from the .NET website: http://www.microsoft.com/net

[7] ThePHPGroup. (n.d.) Retrieved December 9, 2013 from the PHP website: http://php.net/

[8] Python Software Foundation. (n.d.) Retrieved December 9, 2013 from the Python website: http://www.python.org/

[9] Ruby Community (n.d.) Retrieved December 9, 2013 from the Ruby website: https://www.ruby-lang.org/en/