

PRG02 : excitement documentation

=====overview

One of the unique features of smartwatches is the focus on personal data: for example, many smartwatches use their integrated accelerometers to track steps taken over a day, or integrated heart rate monitors to measure when a person is getting good cardiovascular exercise. In the theme of the semester, we'll be using sensors as a way to encourage explorers to document their travels: when a watch senses that its wearer's interest is piqued, it will prompt them to document what they see and mark what interested them.

=====the assignment

You will create an application that spans an Android phone and a smartwatch. The purpose of the assignment is to:

- deepen your understanding of smartwatch interactions across both the watch and phone platform through notifications and touch callbacks
- develop fluency integrating with the camera and other applications on Android
- utilize the Twitter API for storing and retrieving information
- understand the usage of the sensors API

Interaction Flow

(photos by edanley and dirkvorderstrasse on Flickr, and from Google Streetview)

Before an outing in San Francisco, a user launches the excitement documentation app and logs in with their credentials, connecting it to their Twitter account.

Walking around San Francisco wearing their Moto 360 watch and carrying their Android smartphone, the user wanders down [Clarion Alley](#) and is startled by the walls coated in street

art. They jump up and down in excitement. This is detected by the watch, which buzzes on their wrist and presents a prompt to record what they were interested in.



The user taps the notification, which opens a camera. The user takes a picture. Once they accept the picture, it's included in a tweet with the hashtag *#cs160excited*. The user enters a message for the tweet -- "Gosh, I should come to Clarion more often!" -- and sends the message to Twitter.



After the tweet is sent off, a recent photo tweeted by someone else with the hashtag *#cs160excited* is displayed on their watch.



You must submit your **source code**, the **executable**, **screenshots** and a **narrated video**. It is your responsibility to ensure that the executable has all the resources it needs to execute.

=====grading criteria

Up to 40 points will be given if your application compiles, runs, contains the functionality as detailed in the instructions. We will grade the **usability** of your application. This means you may lose points for design problems, even if your application formally fulfills the requirements. Examples of problems that will cost points are: color choices that make the interface hard to read; inappropriate sizing and positioning, overly complicated and onerous sequences of steps required to complete tasks, lack of feedback, etc.

- Phone App (15 points)
 - There is a user interface where the user can authorize the app to post with their Twitter login (4 points)
 - After the receiving a message from the watch, the app launches a camera so the user can record what they see as exciting (4 points)
 - After taking the picture, the user is directed to an interface for writing a tweet (4 points), including:
 - The photo they took is automatically loaded into the tweet
 - The tweet includes the #cs160excited hashtag by default
 - Usability of the phone app (3 points)
- Watch App (25 points)
 - A notification appears when the accelerometer gives volatile readings (8 points)
 - (you may define 'volatile readings' as you think appropriate)
 - The notification includes a button that lets the user tell the handheld to start the camera (7 points)
 - After the user submits the tweet on the handheld app, the watch shows a photo from the most recent tweet with the hashtag #cs160excited that was written by another user (7 points)
 - Usability of the watch app (3 points)
- Video: Submit a narrated video demo of your application, max of 90 seconds, that shows and describes the features.
- Extra credit: Up to 5 extra credit points are available if you design and implement more features, such as:
 - Use speech recognition for launching the camera app
 - Implement location sensitivity: users' photos can be tagged with their location, and the twitter photo retrieved can come from nearby.
 - Any additional feature you feel improves the overall usability (make sure you draw our attention to that in the video you submit)

=====implementation notes

One way you could approach this assignment is the following.

- First, get your watch emulator and handheld emulators talking -- you should be able to see a notification passed from one to the other.

- Then get excitement sensing working on the watch.
- Next, implement communication between the watch and the handheld using messages.
- Finally, you should connect your app to Twitter and implement the mobile user interface.

To get the watch and mobile emulators talking to each other, we recommend a specific setup: we ran a Wear emulator running on API 22 downloaded from the Android SDK Manager, connected to a Genymotion handheld emulator running Android 5.1.0. You need to download the Google Play services apps and Android Wear and install them in the handheld emulator. You may have to restart the emulator several times to get everything installed cleanly. Then you need to set up a gateway between the two emulators with this command:

```
adb -s <handheld-ip>:<handheld-port> -d forward tcp:5601 tcp:5601
```


where you can find the handheld IP and port with the command `adb devices` from a command line that has access to the Android SDK tools. Then you can import a project like the “Wearable Notifications” application in Android Studio to make sure notifications are going through.

Sending notifications from either device is pretty straightforward, and we direct you to [Creating a Notification for Wearables](#) for a quick how-to.

For sensing, you’ll want to fetch the accelerometer sensor from the [SensorManager](#) and register a listener to it to stay updated to its changing values. This should run in the background. It’s up to you to figure out how to do this monitoring without having an application on the wear explicitly showing up on the screen. As you’re working with an emulator, you can [manually update the value of the accelerometer in the emulator using telnet](#).

Messages can be passed from the wearable to the handheld using a `GoogleApiClient` and received using a `WearableListenerService`. The wearable should scan for *nearby nodes* that offer the *capability* of taking a picture and tweeting, and send a message to the nodes with that capability. The handheld needs to *broadcast* this capability in the first place. See the Android doc on [Sending and Syncing Data](#) for information on capabilities and message-passing.

To send messages between the wearable and handheld, both the application on the wear and the handheld have to have the same application ID. To make sure that both have the same application ID, you can create both the wearable and handheld applications together by checking their appropriate boxes when you create a “New Project...” in Android Studio.

Launching a camera for an app is a common activity for many applications, and to do it, you need to start an Intent with the flag `MediaStore.ACTION_IMAGE_CAPTURE`. Make sure to save the image to a known URI -- you will need it to access the photo later. In Genymotion, you can click on this button  to route your computer’s webcam to the emulated cameras.

To integrate Twitter, we recommend you work with the [Fabric API](#). There’s a plugin that works with Android Studio to help instrument your app (though you have to be careful to work with the right version of the Genymotion emulator). Install the Twitter app on the device. Once you have an account with Fabric, you can use their utilities to an app to authorize on the user’s behalf, write tweets, and fetch past tweets and their data.

Finally, this project likely requires at least 2 Android applications (one for Wear and one for handheld) and at least one activity and service for each of these applications. [Activities](#) are usually reserved for graphical components, [services](#) for background work or self-contained computational work, and [intents](#) for starting and passing data to activities or services.

You are free to design your own app provided that it has at least the functionality enumerated above. For best practices designing for Wear (when to use notifications, how to be context aware, etc.), we direct you to the Android guidelines for [Wear](#) and [Android in general](#).

=====screencast

What your screencast should contain:

- Narrated walkthrough of the interface from the user's perspective
- No implementation details
- In the end, point out any extra credit points

Be concise. Your video shouldn't be longer than 90 seconds for the basic functionality. You may use additional time to point out possible extra credit features. Be prepared to do multiple takes; writing a script down first is helpful.

Don't record the entire screen - only record the Android emulator windows.

There are different free software packages available to create narrated screencasts. Here are some options:

- Windows: CamStudio
- OS X: Quicktime X
- Linux: <http://www.linuxhaxor.net/?p=815>
- screencast-o-matic.com

Please crop your video to only show the emulator - we don't need to see your entire desktop.

===== submission instructions

We will be using hackster.io to upload materials in an easy-to-read and visually consistent way. Use this assignment information as a guide to ensure that all the relevant grading criteria can be easily found.

After this assignment is submitted, you will be asked to evaluate two of your peers' apps, following a format similar to the studio in week 1.