CIS 4500 - Special Topics

Wear OS (Android Wearables)

Summary

Wear OS is a valuable extension to your Android ecosystem that provides you with an opportunity to immerse your user. This exercise will cover the essential building blocks to develop a Wear OS application and set up an accompanying complication data provider.

Upon completion of the exercise, you will have a complete complication service and wear OS application that is translatable to your CIS4500 project context.

Implementation

Setting up the Environment

1. Pull the git repo containing the cast and wear exercises: https://github.com/xairos/cast-and-wear-exercises.git
2. Open the `“cast-and-wear-exercises/wear-os/app-exercise” folder` in Android Studio.

Task A - Populating the Demo Application

1. Open the AndroidManifest.xml and find the following within the service declaration:

```
<meta-data
   android:name="android.support.wearable.complications.SUPPORTED_TYPES"
   android:value="TASK_1"/>
<meta-data

android:name="android.support.wearable.complications.UPDATE_PERIOD_SECONDS"
   android:value="TASK_1" />
```

I. Set the value of the `“.SUPPORTED_TYPES”` types to “`SHORT_VALUE`”. This piece of metadata defines the acceptable data types displayed in our complications.

II. Set the value of the "`.UPDATE_PERIOD_SECONDS`" to "`0`". This will trigger updates to the complication upon request.

2. Open "`MainActivity.java`" and review the notes provided in **TASK A-2** and insert the following code block as specified:

```
new ProviderUpdateRequester(this, new ComponentName(this , /*1*/))
```

   a. Replace "`/*1*/`" with "`DemoComplicationProviderService.class`". We are telling the operating system to set up a "`ProviderUpdateRequester`" for our complication in "**`this`**" context.
   b. At the end the code block insert "`.requestUpdateAll();`". This addition allows you to call a method from the "`ProviderUpdateRequester`" to trigger an update

3. Open "`DemoComplicationProviderService.class`" within the provider package and review the notes provided in **TASK A-3**. Upon completion, insert the following code at the "`/* TASK A-3 */`" tag.

```
complicationData = new ComplicationData.Builder(/*2*/)
        ./*3*/(ComplicationText.plainText(complicationValue))
        .build();
```

   a. Change "`/*2*/`" to "`ComplicataData`" proceeded with a "`.`". Afterwards, using Android Studio's autocomplete, observe the possible data type options available.
      i. Which data type makes sense? Consider the AndroidManifest.xml.
   b. Remove "`/*3*/`" and the rest of the line entirely and proceed with "`.set`". Autocomplete the rest of the method with the corresponding appropriate data type and populate the method items again.

Review: At this point, we're transforming our obtained data to be ready for a complication on the watch face.

At this point, you have completed the initial exercise. You have learned the simplicity of implementing a complication data provider.

TASK A - Question Set

1. What are the possible data types available for use in a complication?
2. How does the complication receive data from an application?
3. Does the complication receive the data directly from an application? Explain.
4. What pieces of the android manifest was relevant to the complication service?
5. What pieces of the android manifest were relevant to Wear OS?

TASK B - Stretch Goal

Now it’s time to take your understanding and stretch the demo project. If you’ve noticed when running the project, there is a “LONG” button. Implement the sending “LONG_TYPE” data to a complication whilst still having “SHORT_TYPE” data.

Requirements:

1. Both data types are recorded correctly
2. Both complications receive the appropriate data type

Materials & Resources

- https://developer.android.com/reference/android/support/wearable/complications/ProviderUpdateRequester
- https://developer.android.com/reference/android/support/wearable/complications/ComplicationProviderService
- https://developer.android.com/training/wearables/watch-faces/complications
- https://developer.android.com/training/wearables/watch-faces/exposing-data-complications