Wear Communication Demo

In case you found something to improve, please tell us!
`https://forms.gle/jmjAr3ZRChogeTVq8`

This minimal example illustrates how to use the **`WearService`** provided to send data between an Android device and a Wear device.

1 Usage

This example app has a simple workflow:

1. Start the Wear app, so the listeners are registered for incoming events
2. Start the Mobile app, to send the demo data to the watch

Following this order, the Mobile app sends a **`String`** saying ”Hello Wear” using the **Message** API and a **`Bitmap`** image using the **Data** API.

Tip: You can easily find where is the demo code by searching for ”example” (case insensitive) in the whole project.

2 How it’s built

2.1 Initial setup

We created an android project from scratch with the following steps to create both the mobile and wear modules at the same time:

1. Create a new Android project,
2. In creation wizard, we selected the tab named **Wear OS**, choosing a **Blank Wear Activity**,
3. On the next page, before finishing the wizard, we checked the box named **Pair with Empty Phone app**.

We copied the **`WearService`**`.java` file provided in Moodle along with this handout in the **`mobile`** module. Doing it manually, it must be in the **`mobile/src/main/java/ch/epfl/esl/wearcommunicationdemo`** folder. When doing a drag’n’drop of the file durectly in Android Studio’s file browser on the left, it puts the file in the right place directly.

Then, we made sure to update the package name at the top of the file:

```java
/** mobile -> WearService.java **/
package ch.epfl.esl.wearcommunicationdemo;
```

We registered the **WearService** service in the app's manifest as follows:

```xml
<!-- mobile -> AndroidManifest.xml -> <application ...> -->
<service android:name=".WearService">
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.DATA_CHANGED" />
        <data
            android:host="*"
            android:pathPrefix=""
            android:scheme="wear" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
        <data
            android:host="*"
            android:pathPrefix=""
            android:scheme="wear" />
    </intent-filter>
</service>
```

Then, we edited the **build.gradle** of the **mobile** module to add the following code:

```
/** mobile -> build.gradle -> android{...} -> buildType{...} **/
buildTypes.each {
    project.ext.constants.each {
            // - String constants used in Java as `BuildConfig.W_a_key`
            // - Resources are used as usual:
            //     - in Java as:
            //       `[getApplicationContext().]getString(R.string.W_a_key)`
            //     - in XML as:
            //       `@string/W_a_key`
        k, v ->
            it.buildConfigField 'String', "W_${k}", "\"${v}\""
            it.resValue 'string', "W_${k}", v
    }
}
```

This setup phase was repeated for the **wear** module (Java file, manifest update and **build.gradle** patch).

The final setup step is to change the **project**'s **build.gradle** file to allow the definition of String constants across all projects, preventing typing mistakes and hard-to-debug problems. This is done by adding the following:

```
/** project -> build.gradle -> allprojects{...} **/
// Constants defined for all modules, to avoid typing mistakes
// We use it for communication using the Wear API
// It is a key-value mapping, auto-prefixed with "W_" for convenience
project.ext {
    constants = [
    ]
}
```

2.2 Sending through the Wear APIs

In a first example, we want to send a string from the **mobile** app using the **WearService** to the **wear** app. The Wear API uses internally a Bluetooth communication channel to share data between devices. In the **wear** app, the **WearService** receives the data, and forward it to its **MainActivity** for the user to see the result.

For this example, we defined in the mobile's **WearService.java** two different actions in **enum** called **ACTION_SEND**:

```
/** mobile -> WearService.java -> WearService **/
// Actions defined for the onStartCommand(...)
public enum ACTION_SEND {
    EXAMPLE_SEND_STRING, EXAMPLE_SEND_BITMAP
}
```

These two new action are used in the **onStartCommand(...)** method that is called when calling the service from another place. One action uses the **Message** API, while the other one uses the **Data** API:

```
/** mobile -> WearService.java -> WearService -> onStartCommand(...) **/
case EXAMPLE_SEND_STRING:
    // Example action of sending a String received from the MainActivity
    String message_to_send = intent.getStringExtra(MainActivity
            .EXAMPLE_INTENT_STRING_NAME_ACTIVITY_TO_SERVICE);
```

```
    sendMessage(message_to_send, BuildConfig.W_path_example_message);
    break;
case EXAMPLE_SEND_BITMAP:
    // Example action of sending the app icon, resized to 128 pixels
    Bitmap iconBitmap = BitmapFactory.decodeResource(
            getApplicationContext().getResources(), R.drawable.esl_logo);
    Asset iconAsset = createAssetFromBitmap(iconBitmap, 128);
    // Create the datamap
    PutDataMapRequest putDataMapRequest = PutDataMapRequest
            .create(BuildConfig.W_path_example_datamap);
    putDataMapRequest
            .getDataMap()
            .putAsset(BuildConfig.W_datamap_example_image, iconAsset);
    sendPutDataMapRequest(putDataMapRequest);
    break;
```

The first case **EXAMPLE_SEND_STRING** retrieves a **String** sent as an **Intent**, using the constant defined in the **MainActivity.java** file:

```
/** mobile -> MainActivity.java -> MainActivity **/
public static final String EXAMPLE_INTENT_STRING_NAME_ACTIVITY_TO_SERVICE =
        "EXAMPLE_INTENT_STRING_NAME_ACTIVITY_TO_SERVICE";
```

It then proceeds to send it using the **Message** API, on the **W_path_example_message** path that we defined as a constant in the project's **build.gradle** file:

```
/** project -> build.gradle -> allprojects{...} -> project.ext{...} **/
constants = [
        path_example_message: "/EXAMPLE_PATH_MESSAGE",
]
```

The second case **EXAMPLE_SEND_BITMAP** creates a **Bitmap** object from a resource file (our lab's logo, in **mobile/src/main/res/drawable/esl_logo.png**), and adds it in a **DataMap** under the name **W_datamap_example_image**. The **DataMap** is headed to the path **W_path_example_datamap**. These two new constants are again defined in the project's **build.gradle** file:

```
/** project -> build.gradle -> allprojects{...} -> project.ext{...} **/
constants = [
        ...
```

```
        path_example_datamap: "/EXAMPLE_PATH_DATAMAP",

        datamap_example_image: "EXAMPLE_IMAGE_NAME_IN_DATAMAP",
]
```

If any errors subsist after defining these constants, be sure to build the project so that Android Studio regenerates the corresponding **BuildConfig** class.

Now the **WearService** is ready to operate, we can add code in the **MainActivity** to call the service and send data to the wear device:

This message is sent on the **mobile**'s **onCreate(...)** method:

```
/** mobile -> MainActivity.java -> MainActivity -> onCreate() **/
// Request to send a string
Intent intent_send_string = new Intent(this, WearService.class);
intent_send_string.setAction(WearService.ACTION_SEND.
        EXAMPLE_SEND_STRING.name());
intent_send_string.
putExtra(EXAMPLE_INTENT_STRING_NAME_ACTIVITY_TO_SERVICE, "Hello Wear");
startService(intent_send_string);
// Request to send a bitmap
Intent intent_send_bitmap = new Intent(this, WearService.class);
intent_send_bitmap.setAction(WearService.ACTION_SEND.EXAMPLE_SEND_BITMAP.name());
startService(intent_send_bitmap);
```

2.3 Receiving from the Wear APIs

On the smartwatch, we need to add code to the **WearService.java** in order to handle the events received from the **Message** and **Data** APIs.

2.3.1 The Message API

Receiving from the **Message** API, we created a new **case** in the method **onMessageReceived(...)** as follows:

```
/** wear -> WearService.java -> WearService -> onMessageReceived(...) **/
case BuildConfig.W_path_example_message:
    // The message received is already extracted in the `data` variable
    Intent intent = new Intent(MainActivity
            .EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_MESSAGE_STRING_RECEIVED);
```

```
    intent.putExtra(MainActivity
            .EXAMPLE_INTENT_STRING_NAME_WHEN_BROADCAST, data);
    LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
    break;
```

Using the constant **EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_MESSAGE_STRING_RECEIVED** we define in the **MainActivity**, we create a local broadcast **Intent** that contains the **String** of our message as the extra named **EXAMPLE_INTENT_STRING_NAME_WHEN_BROADCAST**. The constants are declared with the following code:

```
/** wear -> MainActivity.java -> MainActivity **/
public static final String
        EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_MESSAGE_STRING_RECEIVED =
        "EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_MESSAGE_STRING_RECEIVED";
public static final String
        EXAMPLE_INTENT_STRING_NAME_WHEN_BROADCAST =
        "EXAMPLE_INTENT_STRING_NAME_WHEN_BROADCAST";
```

In the **wear**'s **MainActivity**, we created a broadcast receiver that change the **TextView**'s text using the extra **String** from the **Intent** using the same constant as previously:

```
/** wear -> MainActivity.java -> MainActivity **/
private BroadcastReceiver mBroadcastReveiverString = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        mTextView.setText(
                intent.getStringExtra(EXAMPLE_INTENT_STRING_NAME_WHEN_BROADCAST));
    }
};
```

Finally, we need to register and unregister this **BroadcastReceiver** in the **onResume()** and **onPause()**, respectively:

```
/** wear -> MainActivity.java -> MainActivity **/
  @Override
  protected void onResume() {
      super.onResume();

      // Register broadcast from WearService
      LocalBroadcastManager
```

```
            .getInstance(this)
            .registerReceiver(mBroadcastReveiverString, new IntentFilter(
                    EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_MESSAGE_STRING_RECEIVED));
    }

    @Override
    protected void onPause() {
        super.onPause();

        // Un-register broadcast from WearService
        LocalBroadcastManager
                .getInstance(this)
                .unregisterReceiver(mBroadcastReveiverString);
    }
```

At this point, a **String** is sent from the mobile's **MainActivity** to its **WearService**, then to the wear's **WearService**, and finally broadcast to the wear's **MainActivity** to be displayed on the smartwatch's screen!

2.3.2 The Data API

Similarly to the **Message** API, for receiving events from the **Data** API, we created a new **case** in the method **onDataChanged(...)** as follows:

```
/** wear -> WearService.java -> WearService -> onDataChanged(...) **/
case BuildConfig.W_path_example_datamap:
    // Extract the data behind the key you know contains data
    Asset asset = dataMapItem
            .getDataMap()
            .getAsset(BuildConfig.W_datamap_example_image);
    intent = new Intent(MainActivity
            .EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_IMAGE_DATAMAP_RECEIVED);
    decodeAndBroadcastBitmapFromAsset(asset, intent, MainActivity
            .EXAMPLE_INTENT_IMAGE_NAME_WHEN_BROADCAST);
    break;
```

Using the constant **EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_IMAGE_DATAMAP_RECEIVED** defined in the **MainActivity**, we create an **Intent** that will be used for a local broadcast inside the function **decodeAndBroadcastBitmapFromAsset(...)**. The image data is

retrived and send in the broadcast as a byte array under the constant name **EXAMPLE_INTENT_IMAGE_NAME_WHEN_BROADCAST**. The constants are defined with the following code:

```
/** wear -> MainActivity.java -> MainActivity **/
public static final String
        EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_IMAGE_DATAMAP_RECEIVED =
        "EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_IMAGE_DATAMAP_RECEIVED";
public static final String
        EXAMPLE_INTENT_IMAGE_NAME_WHEN_BROADCAST =
        "EXAMPLE_INTENT_IMAGE_NAME_WHEN_BROADCAST";
```

The broadcast receiver for this image data (decoded from the byte array) is defined in **MainActivity**, as follows:

```
/** wear -> MainActivity.java -> MainActivity **/
private BroadcastReceiver mBroadcastReveiverImage = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Retrieve the PNG-compressed image
        byte[] bytes = intent.getByteArrayExtra(
            EXAMPLE_INTENT_IMAGE_NAME_WHEN_BROADCAST);
        Bitmap image = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
        mImageView.setImageBitmap(image);
    }
};
```

The **mImageView** field is just a reference to an **ImageView** item we created in the **activity_main.xml** layout file, that is initialized in the **onCreate(...)** method.

Finally, we need to register and unregister the **BroadcastReceiver**:

```
/** wear -> MainActivity.java -> MainActivity -> onResume() **/
LocalBroadcastManager
        .getInstance(this)
        .registerReceiver(mBroadcastReveiverImage, new IntentFilter(
                EXAMPLE_BROADCAST_NAME_FOR_NOTIFICATION_IMAGE_DATAMAP_RECEIVED));

/** wear -> MainActivity.java -> MainActivity -> onPause() **/
LocalBroadcastManager
        .getInstance(this)
        .unregisterReceiver(mBroadcastReveiverImage);
```

And now, everything is set to receive the image from the mobile to the watch!

3 Troubleshooting

The following steps are helpful to troubleshoot problems of a non-working communication:

- Check if the Bluetooth is on, for both devices
- Check if the tablet and watch are connected together (visible in the WearOS app)
- Install the mobile and wear apps from the same computer (the cryptographic signature needs to match a single developer)
- Check that the package names of the wear and mobile apps are the same
- Be sure that the **`WearService`** is declared in the **`AndroidManifest`**`.xml` of both modules
- Check that you are not confusing system **`Broadcast`**s and **`LocalBroadcast`**s within the app