GaitUp
make sense of motion

Gait Up SA
EPFL Innovation Park, Bat C. CH-1015 Lausanne
www.gaitup.com
TVA CHE-145.136.760

Physilog®5 Android SDK

The Physilog®5 Android SDK allows the creation of new Android applications that use Physilog®5 sensors or the integration of Physilog®5 into existing applications. The Physilog®5 is a Swiss quality movement sensor which measures acceleration, angular velocity and barometric pressure. With the Android SDK, the Physilog®5 can be controlled from your Android application and you can receive the raw data streamed to the Android device. This document contains general information about the SDK as well as details about how to use the available functionalities.

Table of Contents

1. Content of SDK package 3
2. Capabilities 3
3. Technical Information 3
4. Ease of integration 3
5. Using the Android library 4
5.1. Adding the library to an existing project 4
6. Usage of PhysilogLib 5
6.1. Android Permissions 5
6.2. PhysilogManager 5
6.3. Physilog's states 6
6.3.1. Connection State 6
6.3.2. Device state 6
6.4. Physilog(s) Connections and Disconnections 7
6.4.1. AvailablePhysilogsFragment (Pre-made interface) 7
6.4.2. Manual Physilog (dis)connection (See ConnectionFragment.java) 8
6.4.3. OnResume 9
6.4.4. Start and stop the sensor without file transfer during measurement (See RemoteControlFragment.java) 9
6.5. Broadcasts 10

6.5.1. Streaming sensor's data (See SensorFragment.java) 10
6.5.2. File transfer during measurement (See FileFragment.java) 12
6.5.3. File transfer after measurement (See FileTransferFragment.java) 12
6.5.4. List of available broadcasts 13
Contact information 15

1. Content of SDK package

The SDK contains

- A sample Android application that showcases several features of the SDK
- An Android Library in AAR format called "physiloglib-release.aar" which can be found in the libs folder of the sample application
- This User manual for the Android SDK (the Documentation of the library and a short guide on how to add it to a project can also be found in the ReadMe file)

2. Capabilities

The library manages the Bluetooth connections between the Android device and one or multiple Physilog®5. It also provides methods to easily interact with them. With it one can:

- Scan for available Physilog®5
- Manage the connection/disconnection to Physilog®5
- Get the accelerometers values from Physilog®5
- Get the gyroscope values from Physilog®5
- Get the barometric pressure values from Physilog®5
- Get the quaternion values from Physilog®5
- Start a Physilog®5 recording from the Android application
- Transfer the measurements to the application while a recording is in progress
- Stop the recording and finish the transfer of data to a file saved on the Android device

3. Technical Information

The library requires at least Android 5.0 (Android Lollipop). This is because it communicates with Physilog® using Bluetooth Low Energy which was either not present or not stable in previous versions.

Using the library, an application can connect to at most 7 Physilog® devices, but it is recommended to use a maximum of 4-5 devices at a time. More than this can require important resources to receive and manage the data coming from the Physilog®. This is especially important when measurements are sent directly through Bluetooth (streaming mode) and not recorded to a file.

4. Ease of integration

In addition to the examples in the documentation and the sample app, the library also provides a mean to make the integration as quick and pleasant as possible by providing a premade graphical interface for the connection and management of Physilog® devices. This is showcased in the first two parts of the sample app. This is also perfect for quick prototypes that don't need extensive customization or branding of the interface and just want to rapidly have something running.

5. Using the Android library

5.1. Adding the library to an existing project

This is the structure of a classic Android project:

```
|-Yourapp
|--.idea
|--build.gradle <- important
|--gradle
|--app
|----build
|----libs <- add the aar file here
|----jars
|----src
|----build.gradle <- important
```

1. from the root of your project add the physiloglib-release.arr to `app/libs`. Create the `libs` folder if needed.
2. Open the `build.gradle` found in the **root** of your project, and add `flatDir{dirs 'libs'}` like below:

```
allprojects {
    repositories {
        jcenter()

        //Add this
        flatDir {
            dirs 'libs'
        }
    }
}
```

3. Open the `build.gradle` found in the **app** folder of your project, and add the library to your dependencies

```
dependencies {

    //Add this
    compile(name:'physiloglib-release', ext:'aar')
}
```

6. Usage of PhysilogLib

In addition to the explanations given here, five complete examples of interactions are provided in the sample app.

- ConnectionFragment: Do a manual Bluetooth scan and automatically connect to one of the available Physilog® device.
- RemoteControlFragment: Start and stop the recording of the sensor(s) without receiving the data on the Android device.
- SensorFragment: Display the raw data and firmware version from the Physilog® it's connected to.
- FileFragment: Start the recording of measurement in a file and display its transfer to the android device.
- FileTransferFragment: Request the transfer of a specific file previously collected with the sensor.

6.1. Android Permissions

PhysilogLib requires multiple permissions to work, it needs Bluetooth access, localization access (for Bluetooth Low Energy on versions of Android > 5) and storage read/write to store the measurements. Since PhysilogLib is an Android library, it has already declared the needed permissions in its own AndroidManifest.xml. But since Android 6, it is required to dynamically request each permission, so when your first use the `PhysilogManager` in you code, you need to request the permissions:

```
//If you are in an activity and not a fragment, replace both getter by "this"
physilogManager = PhysilogManager.getInstance(getContext());
physilogManager.checkAndRequestPermissions(getActivity());
```

Tip: In Android studio, you can see what the combined manifest will look like by pressing the `Merged Manifest` button on the bottom when editing your manifest.

6.2. PhysilogManager

The state of the library is maintained in a Singleton called `PhysilogManager` that handles the connection with the Physilog®, and gives methods to access them. To get an instance of PhysilogManager use `PhysilogManager.getInstance(context);`

From the PhysilogManager it is then possible to get an instance of the `Physilog` class (or the list of all of them). This object encapsulates the state of a Physilog® device and gives functions that allow to interact with the sensor. Below is a simple example:

```
PhysilogManager pm = PhysilogManage.getInstance(this);
List<Physilog> physilogs = pm.getPhysilogs();

String TAG = "Demo";

//Log some information
```

```
for(Physilog ph: physilogs){
  Log.d(TAG, "name: " + ph.getName());
  Log.d(TAG, "battery: " + ph.getBatteryLevel());
  Log.d(TAG, "state: " + ph.getStateText());
  ...
}

//start streaming of data
for(Physilog ph: physilogs){
  if(ph.getState() == Physilog.STATE_STANDBY_OFF){
    //information transmitted by streaming can be received using broadcast (see next section)
    ph.startStreaming();
  }
}
```

6.3. Physilog's states

A `Physilog` instance has two states. One for the Bluetooth connection (connected, connecting etc.) and one for the actual state of the Physilog® device (standby, recording, streaming etc.).

6.3.1. Connection State

This is the state of the Bluetooth connection between the Android device and the Physilog®. It is defined as a java enumeration and can be accessed through the `getConnectionState()` method of a Physilog instance and can be one of the following:

```
public enum ConnectionState {
 Disconnected, Disconnecting, Connecting, Connected
}
```

Each of the states also has a member variable called `textId` that points to a text in the library's string.xml. You can use this to display or log the connection state.

Tip: Since it's very common to only work on Connected Physilog®, the physilogManager has a function `getConnectedPhysilogs` that directly returns the list of all connected Physilogs.

6.3.2. Device state

This is the state of the Physilog® device, in the sample app you can see it displayed in the list of available Physilog®. It comes directly from the physical device and is represented as a byte. To facilitate its usage, several constants are defined in the Physilog class:

```
class Physilog {

  public final static byte STATE_SET_PHYSILOG_HW;
  public final static byte STATE_STANDBY_OFF;
  public final static byte STATE_PHYSILOG_ON_FILE;
  public final static byte STATE_PHYSILOG_PAUSED;
  public final static byte STATE_USB_MASSE_STORAGE;
  public final static byte STATE_CONFIGURE_PHYSILOG;
```

```
    public final static byte STATE_ENABLE_BLE;
    public final static byte STATE_DISABLE_BLE;
    public final static byte STATE_WORKING_LAST_FILE;
    public final static byte STATE_PHYSILOG_ON_STREAM;

    ....
}
```

As you can see in the sample app's source code, it is usually used to verify that a Physilog® is in Standby state before starting to record / stream. To get a textual representation of this state use the `getStateText()` method from a Physilog object. A quick example on how to use this state:

```
boolean allReady = true;
for(Physilog ph: physilogManager.getConnectedPhysilogs()){
    if(ph.getState() != Physilog.STATE_STANDBY_OFF){
        allReady = false;
        break;
    }
}

if (allReady){
    //Use the physilogs here
}
```

6.4. Physilog(s) Connections and Disconnections

To do the Bluetooth connections between the Android devices and the Physilog®, the library offers you two choices: Use a pre-made interface that handles all the Bluetooth connections and that can be integrated into any activity or displayed on its own. Or use a set of functions to manually start Bluetooth scan, connect to a Physilog® etc. to build your own interface the way you want.

6.4.1. AvailablePhysilogsFragment (Pre-made interface)

This is the pre-made interface. It's an Android Fragment that can be incorporated in any activity or displayed temporarily on the screen to let the user choose to which Physilog® to connect to. It's used in the first 2 examples of the sample app (SensorFragment and FileFragment) If you want to incorporate it into the layout of another activity you can declare it like this:

```
<fragment android:name="com.gaitup.lib.physilog.AvailablePhysilogsFragment"
    android:id="@+id/available_physilogs_fragment"
    android:layout_width="320dp"
    android:layout_height="match_parent"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" />
```

And you don't have to care about anything, it will do its job on its own. The details about how you will be informed of new connections, disconnections etc. follows below.

You can also do more fancy things. For example, if you use the sample app on a phone-sized device, the fragment will not be part of the layout but only appear when a button on the upper right is pressed.

This is done through the use of the Fragment manager. The code for this can be seen in the MainActivity.java.

6.4.2. Manual Physilog (dis)connection (See ConnectionFragment.java)

If you don't want to use the pre-made `AvailablePhysilogsFragment` you can use the methods provided by the `PhysilogManager` to create the same functionalities. PhysilogManager has a function `scanForPhysilogs(...)` that starts a Bluetooth scan for available Physilog®5. Once it's finished it calls a listener of class `AvailblePhysilogListener` and gives it the list of found Physilog®. Let's look at this example:

```java
private Physilog physilog;

@Override
protected void onCreate(Bundle savedInstanceState) {
   ...
   ...
   physilogManager = PhysilogManager.getInstance(getContext());
   physilogManager.checkAndRequestPermissions(getActivity());

   //start searching for physilogs
   physilogManager.scanForPhysilogs(physilogListener);
}


//Called when the scan for Physilogs has finished
private AvailblePhysilogListener physilogListener = new AvailblePhysilogListener() {
   @Override
   public void availablePhysilogs(List<Physilog> physilogs) {

      //List all possible
      for (Physilog ph : physilogs){
         Log.d("Physilog", "Found this available: " + ph.getName())
      }

      //Connect to the first of the list
      if (physilogs.size() > 0){
         physilog = physilogs.get(0);
         physilogManager.connectToPhysilog(physilog.getDevice());
      }

   }
};
```

```
@Override
 public void onDestroy() {
    //Don't forget to disconnect
    if (physilog != null && physilog.getConnectionState() == ConnectionState.Connected){
       physilogManager.disconnectFromPhysilog(physilog.getDevice());
    }
 }
```

As you can see in the example two other functions are useful: `connectToPhysilog(BluetoothDevice)` and `disconnectFromPhysilog(BluetoothDevice)` are to be used for connection and disconnection.

Note: Bluetooth Low Energy devices are typically only discoverable for a short period of time. Therefore, it is possible that between the time of the Bluetooth scan and the call of the function `connectToPhysilog`, the Physilog® has stopped receiving connections, or has already connected with another device. So, you will need to keep track of the connected devices. (Details are given below)

6.4.3. OnResume

In addition to registering in onCreate and unregistering in onDestroy, the Activities and Fragments **must** call the library when onResume is called (see the Activity Life-cycle graph). This allows the library to refresh the connection status with all the Physilogs and re-connnect with the one who lost connection during the application's pause.

```
 @Override
 public void onResume() {
    super.onResume();
    physilogManager.initActivity();
 }
```

6.4.4. Start and stop the sensor without file transfer during measurement (See RemoteControlFragment.java)

The RemoteControl fragment gives the example how the Physilog can be controlled from the application and how event markers can be sent from the application to the Physilog during the measurement using the `sendEventToAllPhysilogs()` method.

You need to register to the broadcasts related to the connection state of the Physilogs. When all connected Physilogs are in Standby_off state it is possible to start a synchronized measurement of the sensors from the application by using `startAllRemoteFiles().`

6.5. Broadcasts

The primary way for the library to send information to the application is through local broadcasts. Activity/Fragments wanting to be informed of changes such as new connections of a Physilog, new sensor's data etc. must register for the broadcast using the code below:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   ...
   ...
   LocalBroadcastManager bm = LocalBroadcastManager.getInstance(this);

   bm.registerReceiver(broadcastReceiverNewPhysilogConnected, new IntentFilter("new_physilog_connected"));
   bm.registerReceiver(broadcastReceiverPhysilogDisconnected, new IntentFilter("physilog_disconnected"));
}

@Override
public void onDestroy() {
   ...
   ...
   LocalBroadcastManager bm = LocalBroadcastManager.getInstance(this);

   bm.unregisterReceiver(broadcastReceiverNewPhysilogConnected);
   bm.unregisterReceiver(broadcastReceiverPhysilogDisconnected);
}
```

Where `broadcastReceiverNewPhysilogConnected` and `broadcastReceiverPhysilogDisconnected` are BroadcastReceiver called when the specified broadcast is triggered. From the intent they receive, they can get the information provided by the broadcast. For example, when a new Physilog® is connected:

```
BroadcastReceiver broadcastReceiverNewPhysilogConnected = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      Bundle b = intent.getExtras();
      final BluetoothDevice device = (BluetoothDevice) b.get("BluetoothDevice");

      Log.d("Physilog", "A new Physilog is connected: " + device.getName());
   }
};
```

A list of all the available local broadcast can be found at the end of this document.

6.5.1. Streaming sensor's data (See SensorFragment.java)

To get the sensor data of a Physilog® in "real time" in the application, it must be in streaming mode. To do this you only need to call the method `startStreaming()` on the Physilog if it is in Standby State (As shown in an example above). A Physilog cannot be streaming and at the same time record to a file, therefore no file is saved on the Physilog® internal memory when it is recording in streaming mode. Once the Physilog® is in streaming mode, the data will automatically arrive on the Android device trough the broadcasts.

There are 2 broadcasts for data streaming:

- new_sensors_data:
    - "BluetoothDevice": BluetoothDevice
    - "SensorsValues": float[8]
        - [0] timestamp
        - [1] gyroX
        - [2] gyroY
        - [3] gyroZ
        - [4] acceleroX
        - [5] acceleroY
        - [6] acceleroZ
        - [7] Baro

  Where the timestamp values are given as index, to obtain timestamp in seconds divide by the highest sampling frequency used in the configuration (f.ex. accelerometer at 128hz, gyroscope at 256hz and barometer at 16hz -> divide by 256).
- new_quaternion_data:
    - "BluetoothDevice": BluetoothDevice
    - "QuaternionValues": float[4]
        - [0] w
        - [1] x
        - [2] y
        - [3] z

This is an example of how the BroadcastReceiver would look for the first broadcast:

```
BroadcastReceiver broadcastReceiverNewSensorsData = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {

    Bundle b = intent.getExtras();
    final BluetoothDevice device = (BluetoothDevice) b.get("BluetoothDevice");
    final float[] values =  b.getFloatArray("SensorsValues");

    float time = values[0];
    float gyroX = values[1];
    float gyroY = values[2];
    ...
};
```

The SensorFragment also illustrates how to obtain information about the sensor settings and its firmware version. This is shown in the method `broadcastReceiverSensorSettingsValue.`

6.5.2. File transfer during measurement (See FileFragment.java)

To have the Physilog® record the measurements to a file and send it over Bluetooth is done through another set of broadcasts. This time the setup is a bit more complicated. Before calling `startAllFilesStreaming()` on the Physilogs, we must first call a method that will prepare the Physilogs for the recording and create a file for the data to go:

```
//Start the recording for every physilog connected
physilogManager.prepareForRecording();

for(Physilog ph: physilogManager.getConnectedPhysilogs()){
   ph.startAllFilesStreaming(); //NEW method name
   //Update the ui
}
```

This time there are 3 broadcasts related:

- file_name:
  - "BluetoothDevice": BluetoothDevice
  - "name": String
- file_stream_progress:
  - "BluetoothDevice": BluetoothDevice
  - "progress": int
- file_generated:
  - "BluetoothDevice": BluetoothDevice
  - "progress": int

The full path of the file (not just the name) can be obtained through the `getFilePath()` method of a Physilog object.

By default, the generated file is recorded on the primary extern storage of the android device in folder called "Physilog". It is possible to choose in which folder they will arrive:

```
//Choose where the files will be recorded
String dir = Environment.getExternalStorageDirectory().toString() + "/Example";
physilogManager.setFilesDirectory(dir);
```

This will not move already existing files previously stored. This will need to be done manually with a file explorer.

6.5.3. File transfer after measurement (See FileTransferFragment.java)

This fragment illustrates how to transfer a file from the Physilog to the Android device after the measurement on the sensor is finished. To start a file transfer, all connected Physilog must be in Standby_off state. The broadcast works similarly to the file transfer during measurement. The difference is that the sensor(s) will not start a new measurement and you need to specify the name of the file you want to receive from the Physilog. This file needs to exist on the sensor's memory.

6.5.4. List of available broadcasts

Here is a list of all available broadcasts and for each of them the extra information they bring:

- new_physilog_connected:
    - "BluetoothDevice": BluetoothDevice
- physilog_disconnected:
    - "BluetoothDevice": BluetoothDevice
- new_sensors_data:
    - "BluetoothDevice": BluetoothDevice
    - "SensorsValues": float[8]
        - [0] <reserved>
        - [1] gyroX
        - [2] gyroY
        - [3] gyroZ
        - [4] acceleroX
        - [5] acceleroY
        - [6] acceleroZ
        - [7] Baro
- new_quaternion_data:
    - "BluetoothDevice": BluetoothDevice
    - "QuaternionValues": float[4]
        - [0] w
        - [1] x
        - [2] y
        - [3] z
- state_changed:
    - "BluetoothDevice": BluetoothDevice
    - "new_state": byte
- state_read:
    - "BluetoothDevice": BluetoothDevice
    - "StateValue": byte
- sensors_settings_value:
    - "BluetoothDevice": BluetoothDevice
    - "AccFreq": int
    - "GyroFreq": int
    - "BaroFreq": int
    - "AccG": int
    - "GyroG": int
    - "Firmware_version" : string

- battery_level:
  - "BluetoothDevice": BluetoothDevice
  - "value": byte // from 0 to 100 and -1 for invalid battery information
- file_name:
  - "BluetoothDevice": BluetoothDevice
  - "name": String
- file_stream_progress:
  - "BluetoothDevice": BluetoothDevice
  - "progress": int //from 0 to 100
- file_generated:
  - "BluetoothDevice": BluetoothDevice
  - "progress": int //Should be 100 since the file is finished

Contact information

At Gait Up, we welcome your feedback and questions.
Please contact us at:
EPFL Innov' Park - C
CH-1015 Lausanne
tel: +41 21 633 7527
mail: contact@gaitup.com
web: www.gaitup.com

Version	Changes	Responsible	Date
1.0.0	Initial release to public with version 1.0.0 of SDK	Marius Rosset / Rebekka Anker	16.10.2017
1.1.0	Changes related to release of v1.1.0 of Android SDK: - New functions with illustration fragments: RemoteControl, FileTransfer - change of method name to start file transfer during measurement	Rebekka Anker	13.02.2018