BUDDY SDK

User Guide v 2.3.10

Date	Version	Auteur	Revision	Commentaire
30-08-2021	V1.0.0	Kevin HOANG	-	-
06-10-2021	V1.1.0	Kevin HOANG		
01-01-2022	V2.0	Kevin HOANG		SDK v2
15-01-2022	V2.1	Kevin HOANG		Tuto
25-01-2022	V2.2	Kevin HOANG		Open app
16-02-2022	V2.2b	Kevin HOANG		Motion detection
23-02-2022	V2.2.1b	Kevin HOANG		Change Voice
14/03/2022	V2.2.2	Mickael COCQUEMPOT		Open camera and display frame
07/04/2022	V2.3	Kevin HOANG		Speech fonctions, and Tracking
15-04-2022	V2.3.1	Kevin HOANG		New dependencies
11-05-2022	V2.3.2	Kevin HOANG		SetView as face
29-06-2022	V2.3.3	Ilya SEMISALOV		App icon
09-08-2022	V2.3.4	Jacques VALENTIN		How to use maven
16-08-2022	V2.3.5	Kevin HOANG		UI
07.09.2022	V2.3.6	Ilya SEMISALOV		UI
19-09-2022	V2.3.7	Kevin HOANG		New install
06-10-2022	V2.3.8	Kevin HOANG		Various fixes
17-10-2022	V2.3.9	Kevin HOANG		Various fixes
24-10-2022	V2.3.10	Kevin HOANG		App parameters

1 -TABLE DES MATIERES

1 - PREREQUISITES :

- Basic knowledge on how to program an Android App
- Android studio > 4.2
- Android SDK >=28
- Graddle plugin >=7.0
- The latest BuddyCore version installed and running on your robot
- ADB installed on your computer
  https://developer.android.com/studio/command-line/adb

2 - CONNECT TO THE ROBOT

The only way to connect to the Robot is over ethernet. To do so you could use a USB-Ethernet adapter or connect over the Wifi network with **adb**.

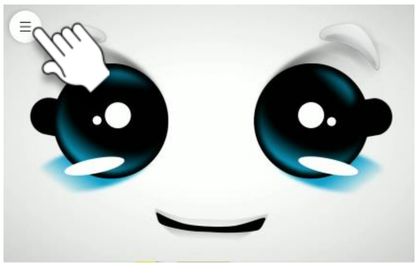
💡 Highly recommended : adb is also provided with **Screencopy,** a useful tool to monitor/control your device from your computer. Go to https://github.com/Genymobile/scrcpy, and install it following the instructions in the "Get the app " section.

On Windows, to connect to the robot with adb:

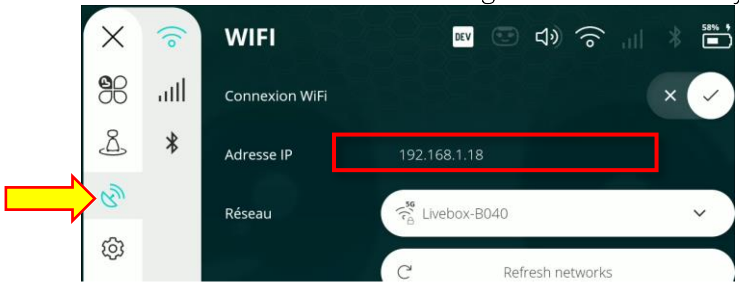
1. Be sure to be on the same wifi network, your PC and Buddy

2. Get the Buddy's IP adress :

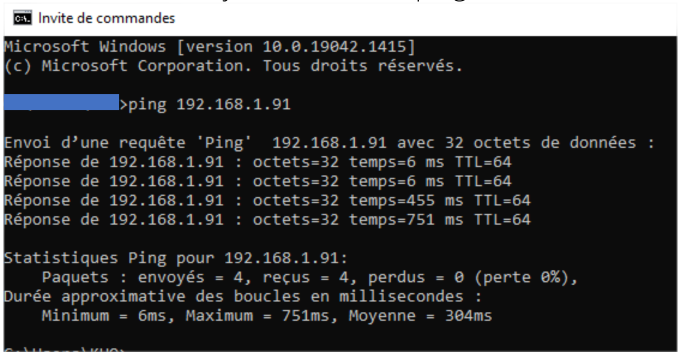
➢ On the robot, access the BuddyCore menu by touching the top-left corner of the screen:



➢ Click on the Connection icon and get the IP address of your robot in the Wifi section



2bis. Check if you can at least ping it

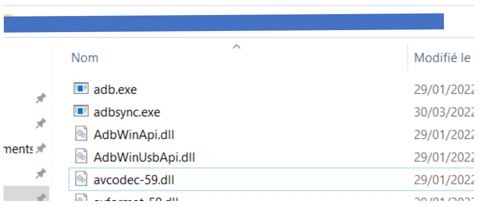


Invite de commandes

```
Microsoft Windows [version 10.0.19042.1415]
(c) Microsoft Corporation. Tous droits réservés.

[redacted]>ping 192.168.1.91

Envoi d'une requête 'Ping'  192.168.1.91 avec 32 octets de données :
Réponse de 192.168.1.91 : octets=32 temps=6 ms TTL=64
Réponse de 192.168.1.91 : octets=32 temps=6 ms TTL=64
Réponse de 192.168.1.91 : octets=32 temps=455 ms TTL=64
Réponse de 192.168.1.91 : octets=32 temps=751 ms TTL=64

Statistiques Ping pour 192.168.1.91:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 6ms, Maximum = 751ms, Moyenne = 304ms
```

On Windows:

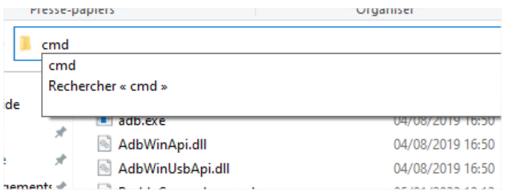
3. Go to a folder containing "adb.exe"



[redacted]

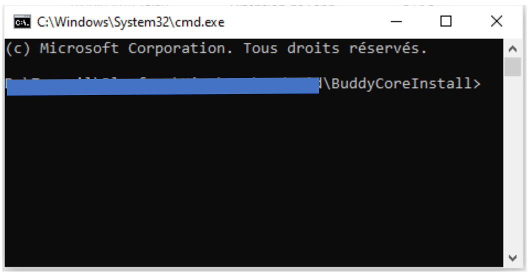
Nom	Modifié le
adb.exe	29/01/202
adbsync.exe	30/03/202
AdbWinApi.dll	29/01/202
AdbWinUsbApi.dll	29/01/202
avcodec-59.dll	29/01/202

4. Open a command invite by clicking on the file bar. Then, type cmd and press ENTER



Presse-papiers
Organiser
cmd
cmd
Rechercher « cmd »
adb.exe
AdbWinApi.dll 04/08/2019 16:50
AdbWinUsbApi.dll 04/08/2019 16:50

5. A command invite should appear like this :

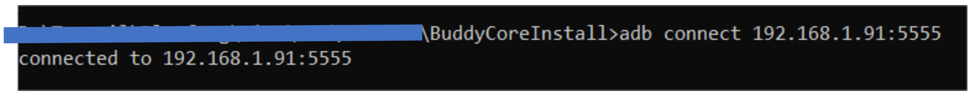


C:\Windows\System32\cmd.exe

```
(c) Microsoft Corporation. Tous droits réservés.

[redacted]\BuddyCoreInstall>
```

6. In This command window, write : " adb connect <robot's Ip Adress>:5555 "

And press ENTER (The Ip Adress is starting like the following form : 192.168.etc…)



```
[redacted]\BuddyCoreInstall>adb connect 192.168.1.91:5555
connected to 192.168.1.91:5555
```

When connected, you should see the message "connected to <IP_Adress> showing up.

3 - YOUR FIRST APP WITH BUDDY

1) Introduction

Your application runs in the Buddy environment, represented by the **BuddyCore** app.

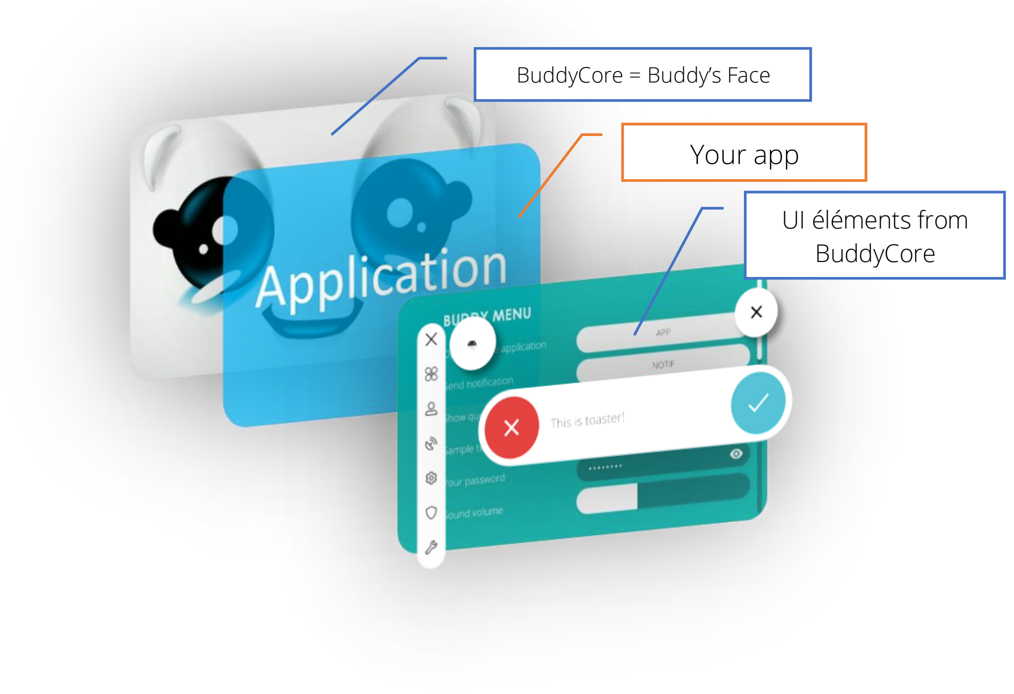
For your app to work properly, BuddyCore must always be launched in the background, so that your app will appear on top of it. That's why, BuddyCore should be the default Launcher on your device.

An important aspect is that the BuddyCore app includes the Face of Buddy. So in fact, if you want to display your robot's face, your app has to be transparent!

In addition, the below instructions and the provided code templates transfer the touch actions on your app to Buddycore, so you can basically interact with Buddy's face when pressing on the touchscreen.

The same way, when you display the menu and UI of BuddyCore, they will appear on top of your app.

For instance, the default close button ✕ and application icon ● are automatically managed by BuddyCore and the SDK, so you don't have to do it yourself.



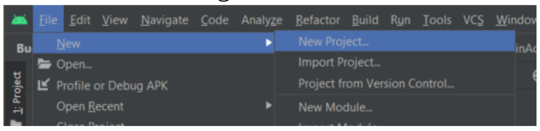
The following explains how to use the BFR SDK within Android Studio from scratch. If you already have an Android project, you can skip directly to the section "Import the SDK module in your project".

For those already familiar with the early versions of the SDK, a changelog can be found here.

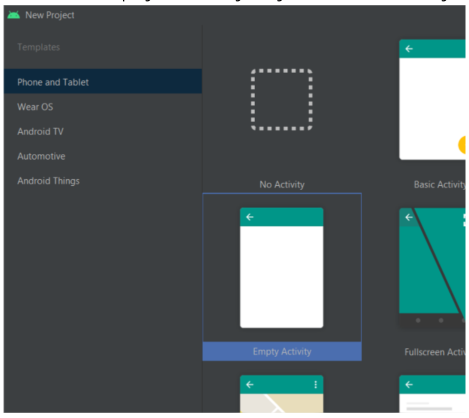
Also, a ready-to-use code template is also provided with the examples.

2) Create an empty Android App

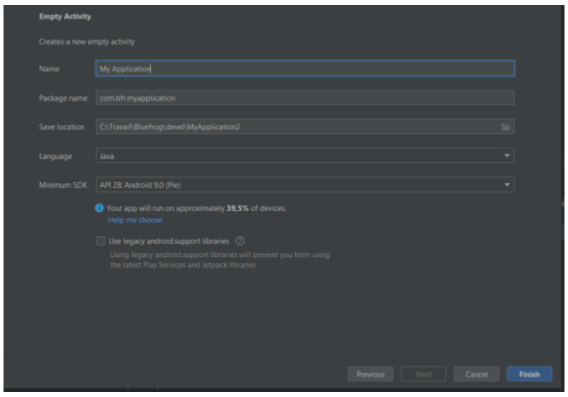
1. In Android studio go to File -> New -> New Project…


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
New
Open…
Profile or Debug APK
Open Recent
New Project…
Import Project…
Project from Version Control…
New Module…

2. Select Empty Activity (by default) but you can change as your conveniance.



3. Name it accordingly , select The API 28 : Android 9.0 as minimum SDK, and click on Finish



Empty Activity

Creates a new empty activity

Name: My Application

Package name: com.bfr.myapplication

Save location: C:\Travail\Bluefrog\devel\MyApplication2

Language: Java

Minimum SDK: API 28: Android 9.0 (Pie)

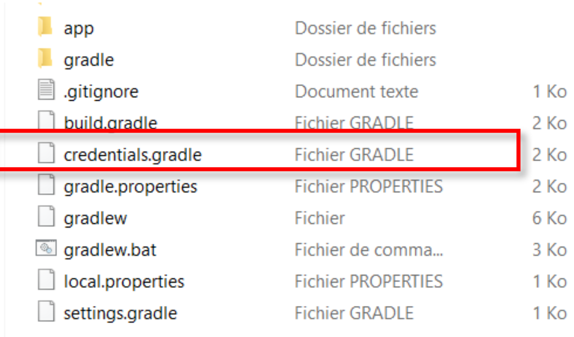
Your app will run on approximately **39,5%** of devices.
Help me choose

☐ Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

3) Import the SDK module and dependencies into your project

1. Create a local file for your Maven credentials

➢ In order to store your username/password for the Maven repository, create a file at the root of your project, and name it "***credentials.gradle***".



app	Dossier de fichiers	
gradle	Dossier de fichiers	
.gitignore	Document texte	1 Ko
build.gradle	Fichier GRADLE	2 Ko
credentials.gradle	Fichier GRADLE	2 Ko
gradle.properties	Fichier PROPERTIES	2 Ko
gradlew	Fichier	6 Ko
gradlew.bat	Fichier de comma...	3 Ko
local.properties	Fichier PROPERTIES	1 Ko
settings.gradle	Fichier GRADLE	1 Ko

➢ In the credentials.gradle file, add the following lines:

```
ext{
    maven_user="<YOUR_USERNAME>"
    maven_password="<YOUR_PASSWORD>"
}
```

➔ Replace <YOUR_USERNAME> and <YOUR_PASSWORD> by the username and password that Bluefrog provided to you during your subscription to the SDK.

⚠: If you use git within your project, be careful not to commit the *credentials.gradle* containing you personal user/password on a public repository!!!

2. Setup the Maven dependency in your project

Step 1: Setup maven repository in your **project** *build.gradle*

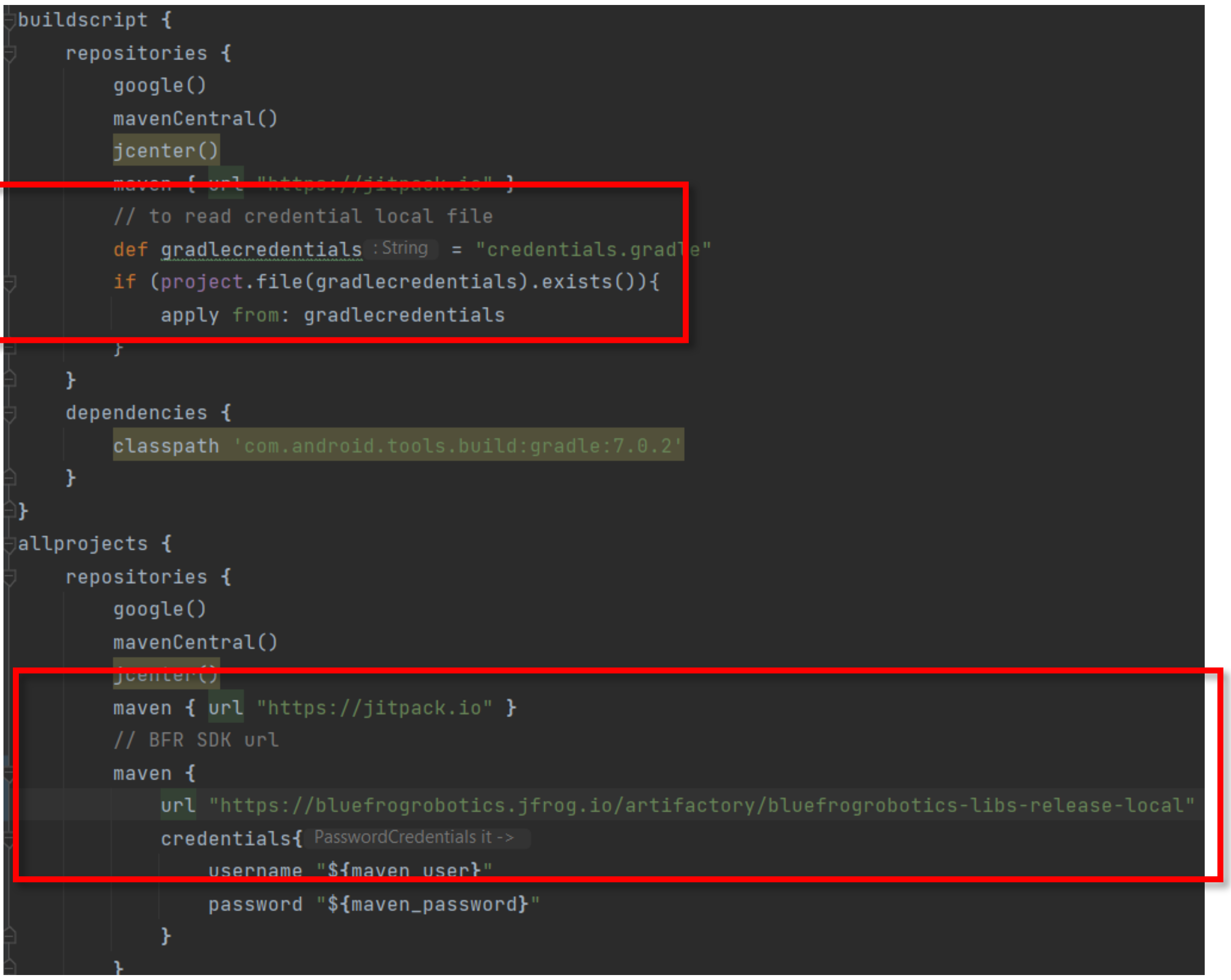
- Go to the **project** *build.gradle* file
- In the buildscript section, add the following lines :

```
def gradlecredentials = "credentials.gradle"
if (project.file(gradlecredentials).exists()){
    apply from: gradlecredentials
}
```

- In the *allprojects/repositories* section, add the following lines for the Maven repository :

```
maven {
url "https://bluefrogrobotics.jfrog.io/artifactory/bluefrogrobotics-libs-release-local/"
     credentials{
          username "${maven_user}"
          password "${maven_password}"
     }
}
```

The graddle file should look like this:



```
buildscript {
    repositories {
        google()
        mavenCentral()
        jcenter()
        maven { url "https://jitpack.io" }
        // to read credential local file
        def gradlecredentials :String  = "credentials.gradle"
        if (project.file(gradlecredentials).exists()){
            apply from: gradlecredentials
        }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.0.2'
    }
}
allprojects {
    repositories {
        google()
        mavenCentral()
        jcenter()
        maven { url "https://jitpack.io" }
        // BFR SDK url
        maven {
            url "https://bluefrogrobotics.jfrog.io/artifactory/bluefrogrobotics-libs-release-local"
            credentials{ PasswordCredentials it ->
                username "${maven_user}"
                password "${maven_password}"
            }
        }
```

Step 2: add the SDK dependencies in the **app** *build.gradle*

- go to the **app** *build.gradle* file
- in the *dependencies* section add the following lines:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'

   // Other dependencies here

    // Bluefrog SDK
    implementation 'com.bluefrogrobotics.buddy:BuddySDK:2+'
}
```

: 'com.bluefrogrobotics.buddy:BuddySDK:2+ means that you import the last version of the Bluefrog SDK for Buddy available. If you need a specific version, you will have to replace 2+ by the exact version number followed by a "+" .
For instance 'com.bluefrogrobotics.buddy:BuddySDK:2.2.0+' , for the SDK v2.2.0.

4) Add the BFR permissions in the Manifest

To use the Buddy *ressources* (Actuators, Motors, Speech, ...), you first need to ask the relevant permissions by adding them in the manifest.

For the purpose of this *hellow_world* example, you don't actually need to add any permission as the app described below doesn't use any resource (other than the screen).

However, for your future developments, you are required to insert one or several of the following lines in the manifest, depending on what your app is supposed to do.

- To access the TTS

```
<uses-permission android:name="com.bfr.buddy.resource.SPEECH" />
```

- To access the STT

```
<uses-permission android:name="com.bfr.buddy.resource.LISTEN" />
```

- To use the wheels

```
<uses-permission android:name="com.bfr.buddy.resource.WHEELS" />
```

- To use the head movements

```
<uses-permission android:name="com.bfr.buddy.resource.HEAD" />
```

- To use the LEDs

```
<uses-permission android:name="com.bfr.buddy.resource.LEDS" />
```

- To access the sensors

```
<uses-permission android:name="com.bfr.buddy.resource.SENSOR_MODULE" />
```

- To change the facial expression

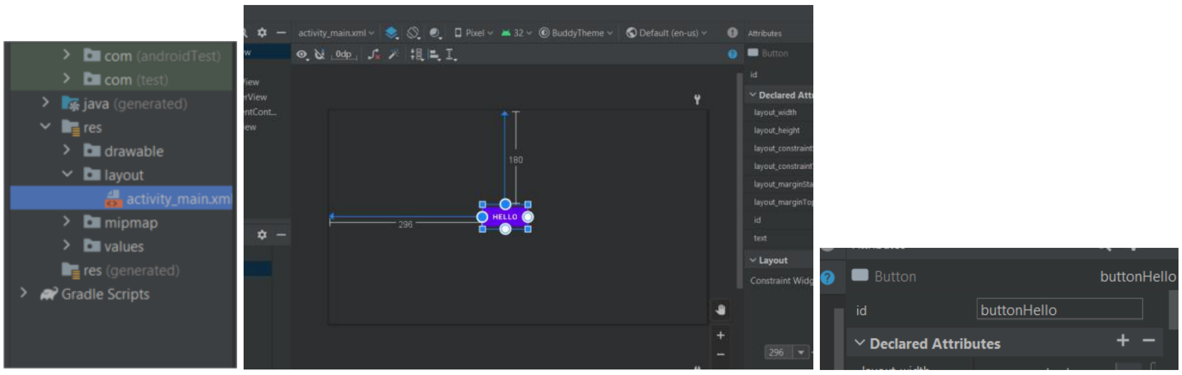
```
<uses-permission android:name="com.bfr.buddy.resource.FACE" />
```

- To use the GUI (made by BFR)

```
<uses-permission android:name="com.bfr.buddy.resource.GUI" />
```

5) Insert a “Hello World” in the layout xml file of your project

Insert a *Hello* button in the layout, and call it buttonHello.



Button buttonHello
id buttonHello

6) Create the main Activity of your app

1. Your MainActivity has to inherit from BuddyActivity.

First, you need to import BuddyActivity

```
import com.bfr.buddysdk.BuddyActivity;
```

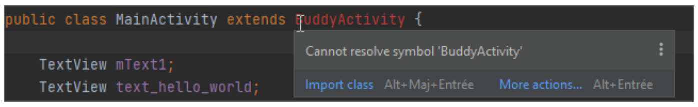
Secondly, you need to extend your MainActivity class with **BuddyActivity** (instead of the default AppCompatActivity)

```
public class MainActivity extends BuddyActivity
```

Tips:

If you do not want to copy/paste. You can follow the following steps.

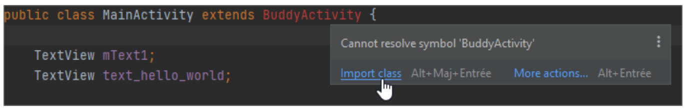
Place your mouse on BuddyActivity



```
public class MainActivity extends BuddyActivity {

    TextView mText1;
    TextView text_hello_world;
```

Cannot resolve symbol 'BuddyActivity'
Import class Alt+Maj+Entrée More actions... Alt+Entrée

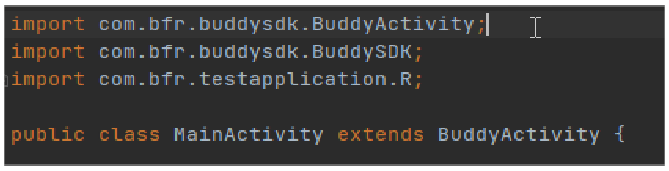
Import class



```
public class MainActivity extends BuddyActivity {

    TextView mText1;
    TextView text_hello_world;
```

Cannot resolve symbol 'BuddyActivity'

Your class has been added



```
import com.bfr.buddysdk.BuddyActivity;
import com.bfr.buddysdk.BuddySDK;
import com.bfr.testapplication.R;

public class MainActivity extends BuddyActivity {
```

2. Inside the MainActivity, two additional functions are needed

   ➢ Add onSDKReady()

```
public void onSDKReady() {    }
```

This callback is called when the BFR SDK is initialized and ready to use.

: As it takes some time for the SDK to initialize, we highly recommend not to make any call to the BFR SDK before onSDKReady is actually called.

3. Send the touch informations to the background (BuddyCore)

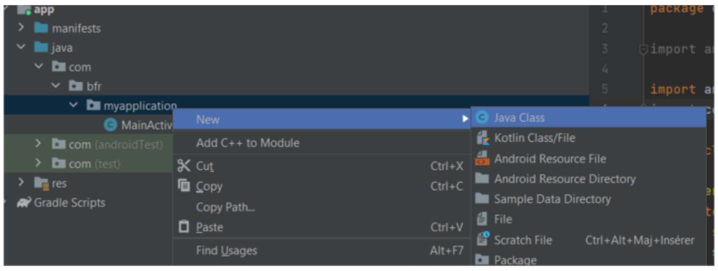
```
// in onSDKReady
BuddySDK.UI.setViewAsFace(findViewById(R.id.view_face));
```

4. Set an onClick listener for the button to make Buddy speak

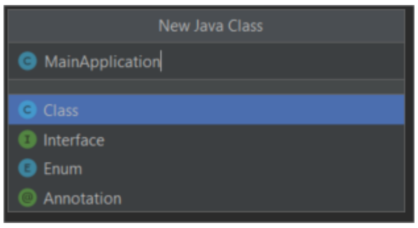
```
// set the button click listener
findViewById(R.id.buttonHello).setOnClickListener(
view -> BuddySDK.Speech.startSpeaking("Hello"));
```

7) Create an application class

1. In the same folder as your MainActivity, create a new class.



2. Call it "MainApplication", for instance



3. The MainApplication class must extends BuddyApplication

```
import com.bfr.buddysdk.BuddyApplication;

public class MainApplication extends BuddyApplication {}
```

4. Check / add if non existent the onCreate callback:

```
public class MainApplication extends BuddyApplication {
    @Override
    public void onCreate()  {
        super.onCreate();
    } //end onCreate

} // end class
```

8) Update the Manifest accordingly

1. Check the MainActivity name

Obviously, the name of the activity in the manifest must match the name of the Main Activity you created in 2-5.

```
<activity
    android:name=".MainActivity"
```

2. Refer to the BuddyApplication

Add an android:name tag, and fill in the name of the application you created in 2-6.

```
<application
    android:name="com.bfr.helloworld.MainApplication"
```

3. Choose a Theme

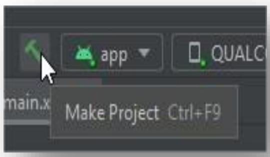
We recommend to use the BuddyTheme provided with the SDK. This theme is for a transparent app as described in "Your first App with Buddy".

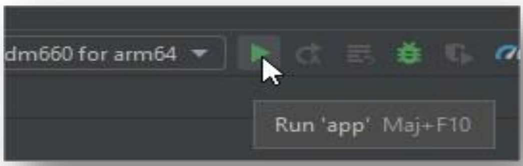
```
<application
      (…)
     android:theme="@style/BuddyTheme">
```

1. At the end, your manifest should look like this:

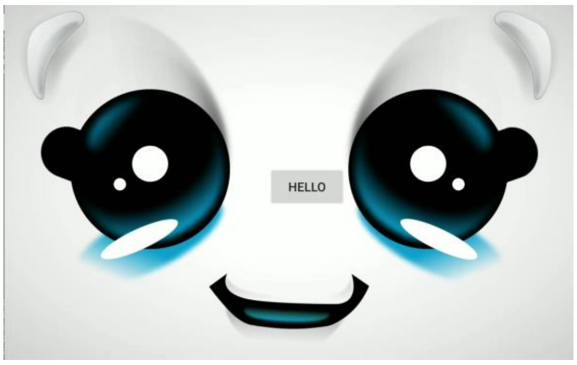
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.myapplication">

    <application
        android:name="com.example.myapplication.MainApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

9) Build and run the app





Your app (a single button in our example) should appear on the screen of the robot:



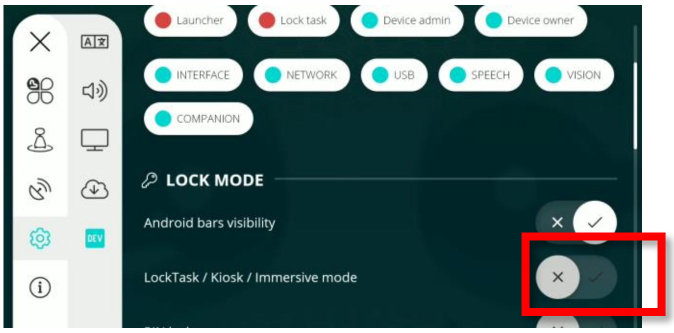
4 - IMPORTANT NOTE ON THE LOCKTASK MODE

By default, your Buddy might be in Locktask, which means:

- Access to the Android default launcher is prohibited
- Only the whitelisted apps can run on you robot
  (for more detailed informations on the subject please refer to the official Android Locktask documentation)

Now, as a developer, you might want to disable the locktask mode for more flexibility in your workflow.

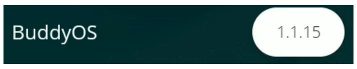
To do so, you can disable the LockTask mode in the *dev* menu from BuddyCore



Android bars visibility ✓

LockTask / Kiosk / Immersive mode ✕

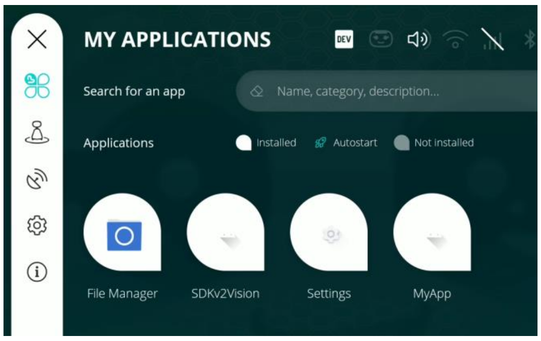
If the *dev* menu is not accessible, you will have to enable it with the following procedure:

- Go to the information menu in BuddyCore
- Press 10x on the BuddyOS version number 
  BuddyOS 1.1.15
- When asked for a pin code, enter the 6 last digits of your robot IMEI and validate
  - ➔ The dev mode should now be active

When in production, we highly recommend to enable the Locktask mode and disable the dev mode. As a consequence, your app will have to be whitelisted if you decide to Locktask your robot.

➔ Please refer to the next chapter to include your app in the whitelist of authorized apps

5 - OPEN AN APP FROM THE BUDDYCORE MENU

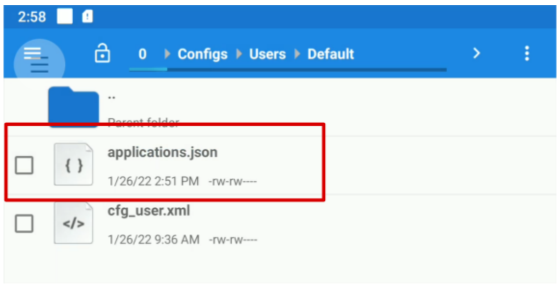


This section explains how to add your app (or any app) to the BuddyCore *My Applications* menu.
This has two main effects:

- You will be able to lauch your app from the menu (obviously)
- The app will be in the whitelist if you want to put your Buddy in Locktask (see previous chapter)

PRE-REQUISITE: You must know the ***PACKAGE name*** *of the app you want to add to the menu. It should look like "com.example.myapp"*

The list of apps in the menu are stored in the *applications.json* file located in /sdcard/Configs/Users/Default/applications.json



Technically, you could edit the file directly on the robot and reboot.

However, we suggest the following method:

1) Download the file on your computer (with a USB stick or the 'adb pull /sdcard/Configs/Users/Default/applications.json <some_folder_in_your_computer> ' command)

2) Manually add the name of the package you want to open at the end of the file like so:

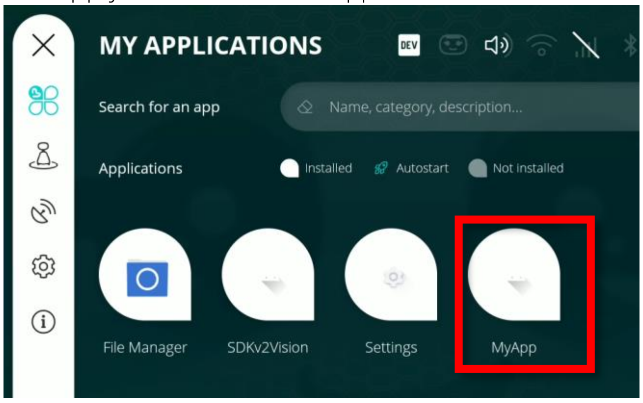
```
 {
  "Autostart": false,
  "Package": "com.android.myapplication",
  "ShowInMenu": true
 }
```

Where :

- Autostart : the app starts automatically with Buddycore
- Package : name of the package of the app
- ShowInMenu : visible or not in the menu
-

3) Save the *applications.json* file back on the device and replace the existing one (with a USB stick or the 'adb push applications.json /sdcard/Configs/Users/Default/applications.json ' command
4) Reboot the robot

The app you added should appear in the menu. Press on the icon to open it.



6 - GUIDELINES FOR A BUDDY APPLICATION

For your app to be integrated in the Buddy general ecosystem (above all the embedded BFR Companion mode), you are required to follow the follow guidelines.

➢ Buddy's Companion mode:
  By default, Buddy shall have an autonomous mode called "Companion". You don't have to do anything really to enable it. You just have to know that your app might run in parallel with other processes.

- Your app must stop its processes when on Pause (onPause of the activity has been called)
- Your app must resume from where it has been on pause (onResume of the activity has been called).
- Your activities must extend BuddyActivity and BuddyCompatActivity instead of Activity and AppCompatActivity
- Do not call any SDK-related function before the onSDKReady() callback has been called.
- The app doesn't need to include a specific button to close as it is already embedded with the SDK
- Because the close and menu button are defined by default, you will have to be careful not to put any sensitive part of UI at this same location in your app.
- You should only request the relevant BFR permissions in the manifest, and leave out the others if you don't use their respective resource.
- For the Graphics guidelines, please refer to the Guideline-Application-UI-2022.pdf document
- Within Android Studio, we recommended to create a custom device when working with layouts. It must be a "phone" of size 1280x800 with only horizontal orientation.

7 - SDK API DOCUMENTATION

1. ACTUATORS

Very often sending a command to the actuators send a return signal via a callback.

⚠: You should wait for the « OK/NOK »signal return before sending another command to the actuators

- HEAD

void USB.enableNoMove(State, RspCallback)

Purpose: Enable the "No" motor

Params:

- State (int): 1 to enable, 0 to disable
- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed
  - "NOK" when fail

string USB.Actuators.getNoStatus()

Purpose: Get the No motor status

Return:

- DISABLE: No motor is disabled
- STOP: No motor is enabled
- SET: No motor is moving
- NONE: Default

void USB.enableYesMove(State, RspCallback)

Purpose: Enable the "Yes" motor

Params:

- State (int) : 1 to enable, 0 to disable
- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed
  - "NOK" when fail

string USB.Actuators.getYesStatus()

Purpose: Get the "Yes" motor status

Return:

- DISABLE: Yes motor is disabled
- STOP: Yes motor is enabled
- SET: Yes motor is moving
- NONE: Default

WARNING:

*You have to **ENABLE THE MOTORS** before using the following functions*

----

void USB.buddySayNo (Speed, Angle, RspCallback)

Purpose: Make the head move around the "No" axis

Params:

- Speed (float): desired speed in °/s between -140 and 140.

  >0: robot is looking right, <0: robot is looking left

- Angle (float) : target angle in ° between -90 and 90°

- RspCallback (IUsbCommadRsp) : return

  - "OK" when succeed to launch
  - "NOK" when fail
  - "NO_MOVE_FINISHED" when the function finished to be executed

void USB.buddySayYes (Speed, Angle, RspCallback)

Purpose: Make the head move around the "Yes" axis

Params:

- Speed (float): desired speed in °/s between -49.2 and 49.2.

  >0: robot is looking up, <0: robot is looking down

  Angle (float): target angle in ° between -35 and 45

- RspCallback (IUsbCommadRsp): return

  - "OK" when succeed to launch
  - "NOK" when fail
  - "YES_MOVE_FINISHED" when the function finished to be executed

void USB.buddySayYesStraight(*Speed, RspCallback*)

Purpose: Make the head move around the "Yes" axis continuously until stop instruction or the physical limit

Params:

- Speed (float): desired speed in °/s between -49.2 and 49.2

  >0: robot is looking up, <0: robot is looking down
- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed to launch
  - "NOK" when fail

void USB.buddySayNoStraight(*Speed, RspCallback*)

Purpose: Make the head move around the "No" axis continuously until stop instruction or the physical limit

Params:

- Speed (float): desired speed in °/s between -140 and 140.

  >0: robot is looking right, <0: robot is looking left
- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed to launch
  - "NOK" when fail

void USB.buddyStopNoMove (*RspCallback*)

Purpose: Stop "No" motor

Params:

- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed
  - "NOK" when fail
  -

void USB.buddyStopYesMove (*RspCallback*)

Purpose: Stop "Yes" motor

Params:

- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed
  - "NOK" when fail

int USB. Actuators.getYesPosition()

Purpose: Get the orientation of the head around the Yes axis

Return:

- Position of the Yes motor in °

int USB. Actuators.getNoPosition()

Purpose: Get the orientation of the head around the No axis
Return:

- Position of the No motor in °

• Wheels

void USB.enableWheels(*Left*, *Right*, *RspCallback*)

Purpose: Enable the Buddy's motors

Params:

- Left (int): enable left wheel (0: Off, 1: On)
- Right (int): enable right wheel (0: Off, 1: On)
- RspCallback (IUsbCommadRsp): return
  - "OK" when launch
  - "NOK" when fail

WARNING:

*You have to **ENABLE WHEELS** before using the following functions.*

----

void USB.rotateBuddy(*Speed*, *Angle*, *RspCallback*)

Purpose: rotate the robot at a given angle and speed

Params:

- Speed (float): give the speed of the rotation of the robot in deg/s around its vertical axis, between -100°/s and 100°/s (min. absolute speed : 30°/s)
  >0: counter-clockwise, <0: clockwise
- Angle[optional] (float): give the angle of the rotation of the wheel in degree,
  - between –360° and 360°
- If absent, Buddy will rotate indefinitely at the given speed
- RspCallback (IUsbCommadRsp): return
  - "OK" when started
  - "WHEEL_MOVE_FINISHED" when the move is finished
  - "NOK" when fail

void USB.emergencyStopMotors(*RspCallback*)

Purpose: Stop the motors immediately with highest possible deceleration

Params:

- RspCallback (IUsbCommadRsp): return
  - "OK" when succeed
  - "NOK" when fail

*void USB.*moveBuddy(Speed, *Distance*, *RspCallback*)

Purpose: Move the robot straight at a defined speed and distance

Params:

Speed (float): give the speed of the robot in m/s, (+): Forward, (-): Backward, between 0.05m/s to 0.7m/s

Distance[optional] (float): give the distance to reach in meter

If absent, Buddy will move indefinitely at the given speed

RspCallback (IUsbCommadRsp): return

- "OK" when launch
- "WHEEL_MOVE_FINISHED" when the move is finished
- "NOK" when fail

void USB.emergencyStopMotors(*RspCallback*)

Purpose: Stop the motors immediately with highest deceleration as possible

Params:

RspCallback (IUsbCommadRsp): return

- "OK" when succeed
- "NOK" when fail

- LEDs

void USB.blinkLed(iLedId, iColor, iPeriod, iRspCallback)

Purpose: make a specific led of Buddy blink with a given color and at a given period

Params:

- iLedId (*int*) :
  - 0 : Right shoulder
  - 1 : Left shoulder
  - 2 : Heart
- iColor (*string*): HTML color (in hexadecimal)
- iPeriod (*int*) : period of the blinking in ms (max 5000ms)
- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.updateAllLed(**iColor**, **iRspCallback**)

Purpose: set the color of all the leds

Params:

- IColor (*string*) : HTML color (in hexadecimal)
- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.fadeAllLed(**iColor**, **iPeriod**, **iRspCallback**)

Purpose: Switch on gradually all leds periodically with the color and period we choose

Params:

- iColor (*string*) : HTML color (in hexadecimal)
- iPeriod (*int*) : period of the blinking in ms (max 5000ms)
- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.updateLedColor(**iLedId**, **iColor**, **iRspCallback**)

Purpose: set the color of a specific led of Buddy

Params:

- iLedId (*int*) :
  - 0 : Right shoulder
  - 1 : Left shoulder

    - 2 : Heart
- iColor (*string*) : HTML color (in hexadecimal)
- iRspCallback (*IUsbCommadRsp*): return
    - "OK" when succeed
    - "NOK" when fail

void USB.updateAllLedWithPattern(iColor, iPattern, iPeriod, iStep, iRspCallback)

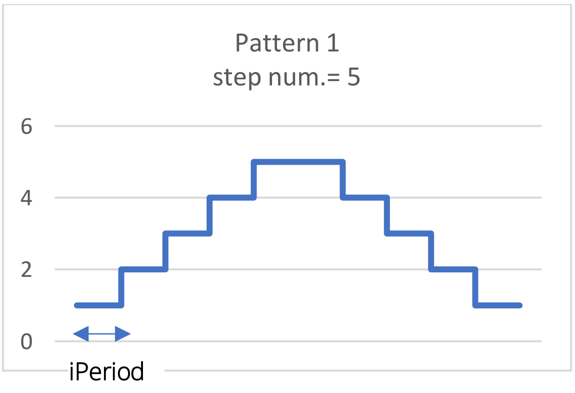
Purpose: Switch all Leds with a pattern with the color we choose.

Params:

- iColor (*string*): HTML color (in hexadecimal)
- iStep (int): number of steps between the OFF and ON value of the LED
- iPeriod (*int*): time interval between each step, in ms (MUST BE >100ms)
- iPattern (*int*) : There are 4 patterns coded on 2 bits (see below)
- iRspCallback (*IUsbCommadRsp*): return
    - "OK" when succeed
    - "NOK" when fail

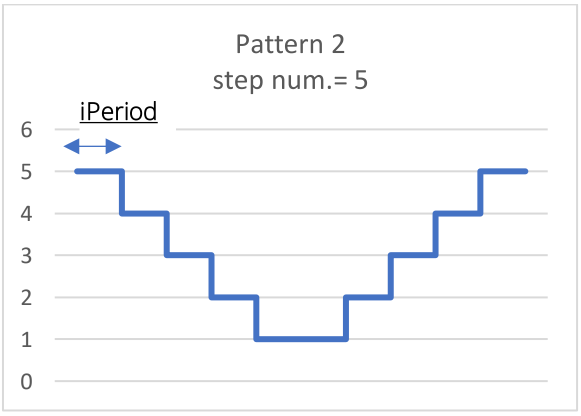
Pattern 1:

The LEDs intensity will increase and decrease following ramps pattern. For a smooth curve, choose a high number of steps.



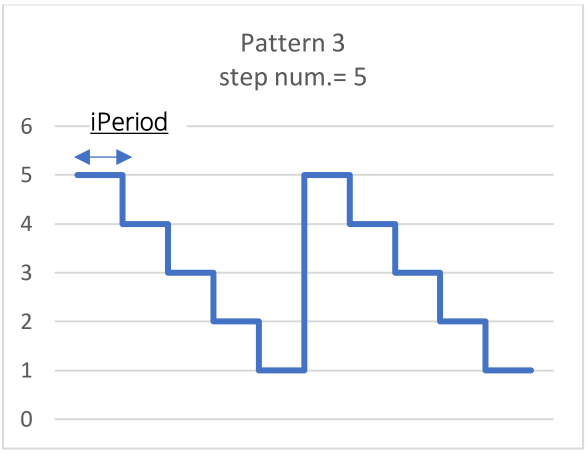
Pattern 2 :

Same as before but beginning with the LED ON.



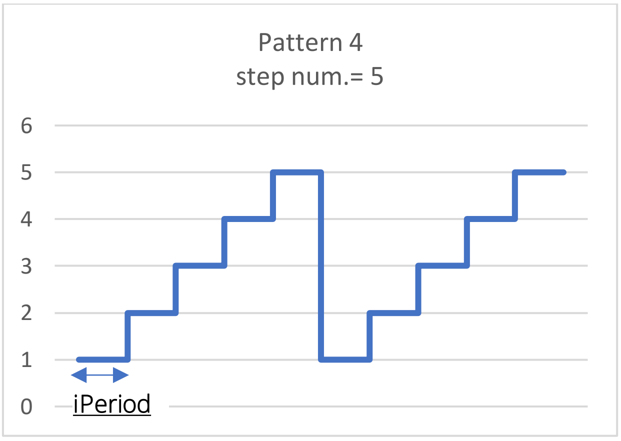
Pattern 3 :

The **LEDs intensity** will decrease following a ramp pattern. For a smooth curve, choose a high number of steps.



Pattern 4 :

The **LEDs intensity** will increase following a ramp pattern. For a smooth curve, choose a high number of steps.



void USB.blinkAllLed(**iColor**, **iPeriod**, **iRspCallback**)

Purpose: make all the leds blink with a specific color at a given period

Params:

- IColor (*string*): HTML color (in hexadecimal)
- IPeriod (*int*) : period of the blinking in ms (max 5000ms)
- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.stopAllLed(**iRspCallback**)

Purpose: Switch off all the Leds

Params:

- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.stopRightShoulderLed(*iRspCallback*)

Purpose: Switch off right shoulder LEDs

Params:

- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.stopLeftShoulderLed(*iRspCallback*)

Purpose: Switch off left shoulder LEDs

Params:

- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

void USB.stopHeartLed(*iRspCallback*)

Purpose: Switch off the heart LEDs

Params:

- iRspCallback (*IUsbCommadRsp*): return
  - "OK" when succeed
  - "NOK" when fail

2. Sensors

• Touch sensors

Buddy has three sensors in the head which can be used to detect a tactile touch. They are located on the top-rear of the head, behind the microphone array, on the right and on the the left.

4. HEAD SENSORS

boolean Sensors.HeadTouchSensors().[Top(), Left(),Right()].isTouched()

Purpose: Check if the top/left/right of the head is touched

Return:

- (*boolean*) true if touched, false if not

For instance:

```
BuddySDK.Sensors.HeadTouchSensors().Top().isTouched();
BuddySDK.Sensors.HeadTouchSensors().Left().isTouched();
BuddySDK.Sensors.HeadTouchSensors().Right().isTouched();
```

5. BODY SENSORS

Buddy has three sensors in the body which can be used to detect a tactile touch. They are located in its torso just above the Bluefrog logo, in its right shoulder and in its left shoulder just above circle of leds.

boolean Sensors.BodyTouchSensor().[Torso(), LeftShoulder(), RightShoulder()] .isTouched()

Purpose: Check if the chest/left shoulder/right shoulder is touched

Return:

- (*boolean*) true if touched, false if not

For instance:

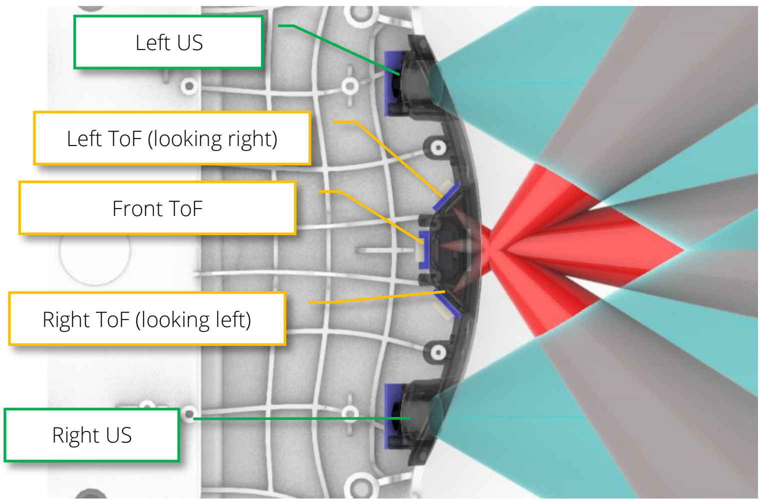
```
BuddySDK.Sensors.BodyTouchSensors().Torso().isTouched();
BuddySDK.Sensors.BodyTouchSensors().LeftShoulder().isTouched();
BuddySDK.Sensors.BodyTouchSensors().RightShoulder().isTouched();
```

Proximity sensors

Buddy has 6 proximity sensors which allow to detect object (or obstacles):

- 1 infra-red Time-of-Flight (ToF) sensor in the back
- 2 ultra-sound (US) sensors in the front, pointing to the left and right of the robot
- 3 ToF sensors in the front: one is in the middle, another one aiming at the Left, the last one aiming at the Right.

⚠: The right ToF sensor actually aims at the left side of Buddy , and the left ToF sensor at the right side of Buddy



The US sensors can give the distance to the first object they see. They return a value in mm between a theoretic range of [0;800mm]

The ToF sensors can give the distance to the first object they see. They return a value in mm between a theoretic range of [0;1300mm]

If there is no object in front of the sensor, it returns 0.

WARNING:

*You have to **ENABLE SENSORS** before using one of the proximity sensors*

----

void USB.enableSensorsModule(*Enable*, *RspCallback*)

Purpose: Enable all sensors

Params:

Enable (boolean): true: enable, false: disable

RspCallback (IUsbCommadRsp): return

- "success" when succeed
- "failed" when fail

6. Ultrasound (US) sensors

int Sensors.USSensors().[RightUS(), LeftUS()].getDistance()

Purpose: Get the right/left US data

Return:

- (*int*) Right US data in mm (returns -1 if the sensor is disabled) between [0;800mm]

For instance:

```
BuddySDK.Sensors.USSensors().LeftUS().getDistance();
BuddySDK.Sensors.USSensors().LeftUS().getDistance();
```

int Sensors.USSensors().[RightUS(), LeftUS()].getAmplitude()

Purpose: Get the right/left US data amplitude. The amplitude is proportional to the size of the object the sensor detects.

Return:

- (*int*) Right US data amplitude in mm (returns 65535 if the sensor is disabled)

For instance:

```
BuddySDK.Sensors.USSensors().LeftUS().getAmplitude();
BuddySDK.Sensors.USSensors().LeftUS().getAmplitude();
```

7. Infra-red Time-of-Flight (ToF) sensors

int Sensors.TofSensors().[FrontMiddle(),FrontLeft(), FrontRight(), Back()].getDistance()

Purpose: Get the distance of the object in front of the respective ToF sensor

Return:

- (*int*) distance of the object in mm (returns 65535 if the sensor is disabled) between [0;1300mm]

For instance:

```
BuddySDK.Sensors.TofSensors().FrontMiddle().getDistance();
BuddySDK.Sensors.TofSensors().FrontLeft().getDistance();
BuddySDK.Sensors.TofSensors().FrontRight().getDistance();
BuddySDK.Sensors.TofSensors().Back().getDistance();
```

boolean Sensors.TofSensors().[FrontMiddle(),FrontLeft(), FrontRight(), Back()].getError()

Purpose: Get the error of the Tof

Return:

- (*boolean*) error of the Tof

- Inertial (IMU) sensors

There are two IMU sensors that gives the linear accelerations and the angular velocity of the robot

They are situated in the body and in the head of the robot.

8. Body IMU

int Sensors.BodyIMU().getAccX()

Purpose: Get the acceleration of the body IMU on the X axis

Return:

- (*int*) value of the X-axis acceleration in mg (0.001 g-force)

int Sensors.BodyIMU().getAccY()

Purpose: Get the acceleration of the body IMU on the Y axis

Return:

- (*int*) value of the Y-axis acceleration in mg (0.001 g-force)

int Sensors.BodyIMU().getAccZ()

Purpose: Get the acceleration of the body IMU on the Z axis

Return:

- (*int*) value of the Z-axis acceleration in mg (0.001 g-force)

int Sensors.BodyIMU().getGyrX()

Purpose: Get the angular velocity of the body IMU on the X axis

Return:

- (*int*) value of the X-axis angular velocity in ddeg/s (0.1degree/second)

int Sensors.BodyIMU().getGyrY()

Purpose: Get the angular velocity of the body IMU on the Y axis

Return:

- (*int*) value of the Y-axis angular velocity in ddeg/s (0.1degree/second)

int Sensors.BodyIMU().getGyrZ()

Purpose: Get the angular velocity of the body IMU on the Z axis

Return:

- (*int*) value of the Z-axis angular velocity in ddeg/s (0.1degree/second)

➢ Head IMU

int Sensors.HeadIMU().getAccX()

Purpose: Get the acceleration of the head IMU on the X axis
Return:

- (*int*) value of the X-axis acceleration in mg (0.001 g-force)

int Sensors.HeadIMU().getAccY()

Purpose: Get the acceleration of the head IMU on the Y axis
Return:

- (*int*) value of the Y-axis acceleration in mg (0.001 g-force)

int Sensors.HeadIMU().getAccZ()

Purpose: Get the acceleration of the head IMU on the Z axis
Return:

- (*int*) value of the Z-axis acceleration in mg (0.001 g-force)

int Sensors.HeadIMU().getGyrX()

Purpose: Get the angular velocity of the head IMU on the X axis
Return:

- (*int*) value of the X-axis angular velocity in ddeg/s (0.1degree/second)

int Sensors.HeadIMU().getGyrY()

Purpose: Get the angular velocity of the head IMU on the Y axis
Return:

- (*int*) value of the Y-axis angular velocity in ddeg/s (0.1degree/second)

int Sensors.HeadIMU().getGyrZ()

Purpose: Get the angular velocity of the head IMU on the Z axis
Return:

- (*int*) value of the Z-axis angular velocity in ddeg/s (0.1degree/second)

• Misc info

➢ Microphone

float Sensors.Microphone().getAmbiantSound()

Purpose: Get the volume in decibel

Return:

- (*float*) the volume in decibel

float Sensors.Microphone().getSoundLocalisation()

Purpose: Get the angle in degree of the sound location between 0° and 360°

Return:

- (*float*) the angle of the sound location in °

float Sensors.Microphone().getTriggerScore()

Purpose: Get the score which rates the pronounciation of the trigger sentence: "OK Buddy"

Return:

- (*float*) the score of the trigger sentence

➢ Battery

int Sensors.Battery().getBatteryLevel()

Purpose: Get the battery level

Return:

- (*int*) the battery level

*bool*ean Sensors.Battery().isCharging()

Purpose: Check if the battery is in charge

Return:

- (*boolean*) true: charging, false: not in charge

3. FACE

⚠ : setMood() uses the face and the LEDs, so it needs two permissions to work :

"com.bfr.buddy.resource.FACE" and "com.bfr.buddy.resource.LEDS"

void UI.setMood(iExpression, *iSpeed, IUIFaceAnimationCallback iCallback*)

Purpose: Give Buddy different facial expressions and set the LEDs
Params:

- iExpression (FacialExpression):
    - NONE
    - NEUTRAL
    - GRUMPY
    - HAPPY
    - ANGRY
    - LISTENING
    - LOVE
    - SAD
    - SCARED
    - SICK
    - SURPRISED
    - THINKING
    - TIRED

- iSpeed[optional] (double): Can take value from 0.0 to 1.0 (0% to 100%). The faster the speed is, the faster the facial expression is.
- iCallback[optional] : Called at the end of the instruction in case of success/failure

💡 : The setFacialExpression() method only controls Buddy's face appearance. Therefore we recommend to use the setMood() method, which also set the LED colors accordingly.

void UI.setFacialExpression(iExpression, iSpeed , IUIFaceAnimationCallback iCallback)

Purpose: Give to Buddy different facial expressions
Params:

- iExpression (FacialExpression):
    - NONE
    - NEUTRAL
    - GRUMPY
    - HAPPY
    - ANGRY
    - LISTENING

- LOVE
- SAD
- SCARED
- SICK
- SURPRISED
- THINKING
- TIRED

- iSpeed[optional] (double): Can take value from 0.0 to 1.0 (0% to 100%). The faster the speed is, the faster the facial expression is.
- iCallback[optional] : Called at the end of the instruction in case of success/failure

void UI.playFacialEvent(iEvent, iSpeed, IUIFaceAnimationCallback iCallback)

Purpose: Play different gimmicks

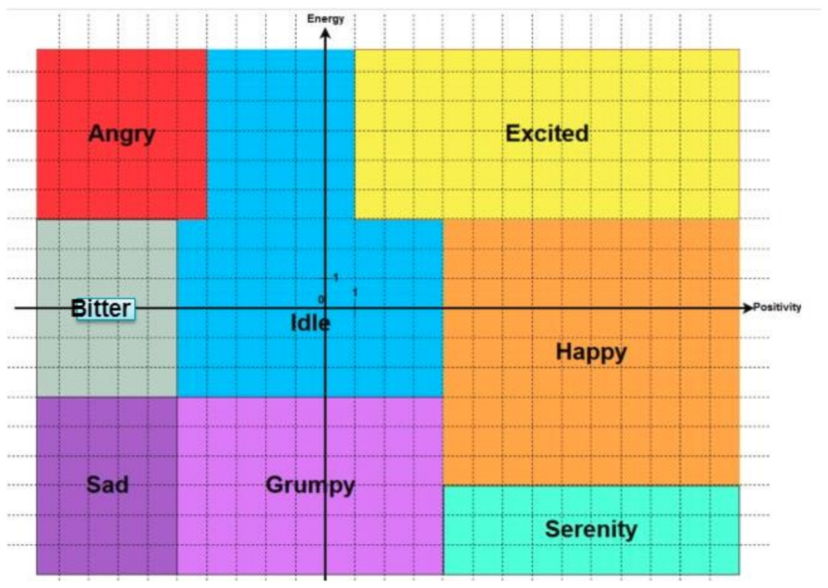
Params:

- iEvent (FacialEvent):
  - DOUBTFUL
  - AWAKE
  - BLINK_EYES
  - BLINK_LEFT_EYE
  - BLINK_RIGHT_EYE
  - CLOSE_EYES
  - CLOSE_LEFT_EYE
  - CLOSE_RIGHT_EYE
  - FALL ASLEEP
  - GROWLING
  - OPEN_EYES
  - OPEN_LEFT_EYE
  - OPEN_RIGHT_EYE
  - SMILE
  - SURPRISED
  - SUSPICIOUS
  - TEASE
  - WHAT
  - WHISTLE
  - YAWN

- Speed[optional] (double): Can take value from 0.0 to 1.0 (0% to 100%). The faster the speed is, the faster the facial expression is.
- iCallback[optional] : Called at the end of the instruction in case of success/failure

➢ Energy/Positivity and facial expressions

When Buddy has a NEUTRAL face, you can fine-tune its facial expression using the Energy/positivity parameters. You will find below a mapping of the possible emotions based on a model from James Russell[1]



void UI.setFacePositivity(***iPositivity***)

Purpose: Change positivity level of the face

Params:

- iPositivity (float): Can take value from 0.0 to 1.0 (0% to 100%).

void UI.setFaceEnergy(***iEnergy***)

Purpose: Change energy level of the face

Params:

- iEnergy (float): Can take value 0.0 to 1.0 (0% to 100%).

void UI.setLabialExpression(***iExpression***)

Purpose: Change the expression of Buddy's mouth

Params:

- iExpression (LabialExpression):
  - SPEAK_ANGRY
  - NO FACE
  - SPEAK HAPPY
  - SPEAK NEUTRAL

[1] The circumplex model of affect : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2367156/

void UI.playFacialRelativeEvent(*iSpeed*)

Purpose: Play face animation relative to the current expression

Params:

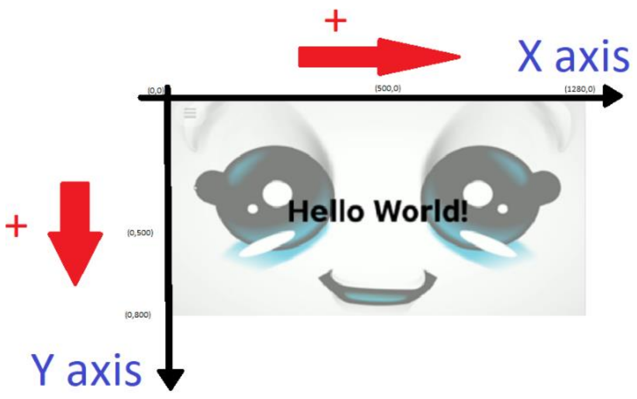
- ISpeed (double): Can take value 0.0 to 1.0 (0% to 100%). The faster the speed is, the faster the gimmick is.

void UI.lookAtXY(*iX, iY, iSmooth*)

Purpose: Make eyes look at a direction

Params:

- iX (float): Horizontal axis. When we go to right, the value of x increases.
- iY (float): Vertical axis : When we go down, the value of y increases.
- ISmooth (boolean): If true the animation will be smooth, otherwise the eyes will instantly move to the direction



void UI.lookAt(iPosition, iSmooth)

Purpose: Make eyes look at a direction

Params:

- iPosition (GazePosition):
  - CENTER
  - TOP
  - LEFT
  - RIGHT
  - BOTTOM
  - TOP_LEFT
  - TOP_RIGHT
  - BOTTOM_RIGHT

    - BOTTOM_LEFT

- iSmooth (boolean): If true the animation will be smooth, otherwise the eyes will instantly move to the direction

void addFaceTouchListener(IUIFaceTouchCallback iCallback)

- Brief : Bind callback to Buddy's face touch event
- Param :
    - IUIFaceTouchCallback iCallback : callback called when face is touched (see IUIFace-TouchCallback)

💡: The Left and Right eyes are actually from **Buddy's** side (not the way you look at it)

void removeFaceTouchListener(IUIFaceTouchCallback iCallback)

- Brief : Unbind callback that was bound to Buddy's face touch event
- Param :
    - IUIFaceTouchCallback iCallback : callback you want to unbind from Buddy's face touch event

4. VOCAL INTERACTION

• SPEECH (Text-to-speech TTS)

For TTS, the SDK uses the ReadSpeaker API.

void Speech.loadReadSpeaker()

Purpose: Initializes ReadSpeaker module. (downloads a license file if not present)

void Speech.startSpeaking(iText, iExpression, iCallback)

Purpose: Says provided text and make a special mouth movement
Params:

- iText (string): Text to speak
- IExpression (LabialExpression):
  - SPEAK_ANGRY
  - NO FACE
  - SPEAK HAPPY
  - SPEAK NEUTRAL
- iCallback: (*ITTSCallback.Stub()*): Callback on success, pause, resume or error. Either onSuccess or onError are called once at the end of the speech. onPause and onResume are called each time a pause (following this pattern \pause=<time_in_milliseconds>\ in iText) starts or finishes.

Note :
It is possible to add pauses in a text to say. To do it, simply add inside the iText string this marker \pause=<time_in_milliseconds>\ . Replace <time_in_milliseconds> by the number of milliseconds you want Buddy to stop his speech.

Warning:
If you want to write it in a Java string you need to escape the \ character by repeating it. So for example write \\pause=1200\\ if you want a pause of 1200 milliseconds.

For ex:

```
BuddySDK. Speech.startSpeaking("Hello \\pause=1200\\ My name is buddy");
```

void Speech.stopSpeaking()

Purpose: Stop speaking

void Speech.setSpeakerPitch(*iPitch*)

Purpose: Sets pitch of speaker

Params:

- iPitch (int): can take value 50% to 200%. The default value is 100%. The lower the pitch is, the deeper the voice is.

void Speech.setSpeakerSpeed(*iSpeed*)

Purpose: Sets speed of speaker

Params:

- iSpeed (int): Can take value 50% to 400%. The default value is 100%.

void Speech.setSpeakerVolume(*iVolume*)

Purpose: Control the volume of speech

Params:

- iVolume (int): Can take value 0% to 300%. The default value is 100%. The higher the value is, the higher the volume is.

int Speech.getSpeakerPitch()

Purpose: gets pitch value

Return:

- (*int*) : pitch value

int Speech.getSpeakerSpeed()

Purpose: get speaker speed

Return:

- (*int*) : speed value

int Speech.getSpeakerVolume()

Purpose: get speaker volume value

Return:

- (*int*) : volume value

boolean Speech.isSpeaking()

Purpose: check if ReadSpeaker is busy and is speaking

Return:

- (*boolean*) : "true" if Buddy is speaking and "false" if not

boolean Speech.isReadyToSpeak()

Purpose: check if ReadSpeaker is fully initialized and is not busy

Return:

- "true" if Buddy is ready to speak and "false" if not

void Speech.setSpeakerVoice(*SpeakerName*)

Purpose: change the speaker voice. The available speakers are:

- "roxane" : French female voice
- "kate" : English female voice

• LISTENING (Speech-to-text STT)

For STT, the SDK uses two external APIs :

- Cerence: an Automatic Speech Recognition (ASR) with two variants:
  - Cerence with grammars: recognizes predefined sentences stored in compiled 'grammar' files. It works offline and returns the recognized sentence (Utterance) as well as an intention (Rule) and score of recognition.
  - Cerence FreeSpeech : recognizes any sentence, in French and English only.
- Google Speech-to-text API : an Automatic Speech Recognition (ASR), that recognizes any sentence, but needs an Internet Connection. It returns only the returns the recognized sentence (Utterance) as well as the score of recognition.

	Works offline	Free speech	Customizable	Languages
Cerence	Yes	No	Yes	FR, EN
Cerence FreeSpeech	Yes	Yes	No	FR, EN
Google FreeSpeech	No	Yes	No	Multi-Lang.

• API of Speech module

STTTask Speech.createCerenceFreeSpeechTask(iLocale)

Purpose: Create a Cerence free speech Speech to Text (STT).

Parameters:

- iLocale (Locale): Locale used (only English and French are supported).

Return: STTTask to manage the STT.
Throw: If an error occurs.

STTTask Speech.createCerenceTask(iLocale, iFcfFilename)

Purpose: Create a Cerence Speech to Text (STT) from a compiled bnf (called a .fcf).

Parameters:

- iLocale (Locale): Locale used (only English and French are supported).
- IfcfFilename (String): Fcf file full path.

Return: STTTask to manage the STT.
Throw: If an error occurs.

STTTask Speech.createCerenceTaskFromAssets(iLocale, iFcfFilename, iAssetManager)

Purpose: Create a Cerence Speech to Text (STT) from a compiled bnf (called a .fcf) located in the assets.

Parameters:

- iLocale (Locale): Locale used (only English and French are supported).
- iFcfFilename (String): Fcf file path from the root of the assets.
- iAssetManager (AssetManager): Asset manager used to have the root of the iFcfFilename.

Return: STTTask to manage the STT.
Throw: If an error occurs.

STTTask Speech.createGoogleSTTTask(iLocale)

Purpose: Create a Google free speech Speech to Text (STT).

Parameters:

- iLocale (Locale): Locale used (only English and French are supported).
    - CANADA
    - CANADA_FRENCH
    - CHINA
    - CHINESE
    - ENGLISH
    - FRANCE
    - FRENCH
    - GERMAN
    - GERMANY
    - ITALIAN
    - ITALY
    - JAPAN
    - JAPANESE
    - KOREA
    - KOREAN
    - SIMPLIFIED_CHINESE
    - TAIWAN
    - TRADITIONAL_CHINESE
    - UK
    - US

Return: STTTask to manage the STT.
Throw: If an error occurs.

- ## *API of STTTask*

void STTTask.start(continuously, iCallback)

Purpose: Start the listening.
Warning: Only one STTTask can be active at a time, so this start will uninitialize any of STTTask objects already initialized.

Parameters:

- continuously (boolean): True for never stoping automatically, false to stop automatically when something is heard.
- iCallback (ISTTCallback): return:

In the type STTResultsData which is an array, we can find the different STTResult that are the different possibilities of what we can say to Buddy.

STTResult hold 3 values:

- getConfidence() : The score of recognition
- getUtterance() : The sentence that the user actually said
- getRule() : The Cerence rule if Cerence engine is used, an empty string otherwise.

void STTTask.pause()

Purpose: Pause the listening.
The robot will not listen anymore but some stuffs will be kept in memory to be able to start again quickly.

void STTTask.stop()

Purpose: Stop the listening.
The robot will not listen anymore and everything will be released.
If you call start after it may be a little slow, to make it kicker you can consider 2 things:

1. Use pause() function instead.
2. Call initialize() function before to call start again().

Advanced functions

void STTTask.initialize()

Purpose: Initialize the listening object.
It is never mandatory, the benefits is that if you call start() after the start functionality will start quicker.
Warning: Only one STTTask can be active at a time, so this initialization will uninitialize any of STTTask objects already initialized.

void STTTask.isRunning()

Purpose: Know if the task is listening.

STTType STTTask.getEngineType()

Purpose: Get the STT engine used by this object.

Return: Values can be:

- GOOGLE_STT
- CERENCE_FREESPEECH
- CERENCE

Locale STTTask.getLocale()

Purpose: Get the locale used by this object.

void STTTask.subscribeToLifecycle(iSTTTaskLifecycleCallback)

Purpose: Subscribe to some events of this object.
Parameters:

- iSTTTaskLifecycleCallback (ISTTTaskLifecycleCallback): Subscription object.

void STTTask.unsubscribeToLifecycle(iSTTTaskLifecycleCallback)

Purpose: Unsubscribe to some events of this object.
Parameters:
iSTTTaskLifecycleCallback (ISTTTaskLifecycleCallback): Subscription object

5. Vision

The SDK Vision module gives access to ready-to use computer vision (CV) algorithms. There are manly two types of algorithms: one-shot and continuous. One-shot algorithms (mainly object detection) are called once, and return almost immediately the result of the CV algorithm from the camera input.

Continuous algorithms work on a continuous stream, and it will consume CPU power while active. So the basic workflow would be:

- Manually start the process
- Get the on-demand information from the respective algorithm
- Manually stop the process when no more needed

By default the camera is started automatically at the opening of your app. However, you could accidentally call for a CV algorithm after the camera has been intentionally stopped and not restarted.

All the algorithms will throw an IllegalStateException("NOT STARTED") if the camera or the continuous algorithm is not started. So you might want to check the status of the camera with the getStatus() method.

General camera functionalities

Vision.startCamera(cameraId, VisionCbk)

Purpose: start the desired camera. By default the first camera (Wide angle) is already started at the beginning, so you won't need to call this function, unless you stopped it yourself.

Params :

- cameraId : the Id of the camera you want to start :
  - 0 : Wide Angle camera
  - ~~1 : Zoom Camera~~ NOT SUPPORTED YET
- VisionCbk : IVision.Stub() that returns:
  - onSuccess : string
  - onError: String

Vision.stopCamera(cameraId, VisionCbk)

Purpose: stop the desired camera. It will unsubscribe the Vision service from the camera, so that you can use it safely from another process.

Params :

- cameraId : the Id of the camera you want to start :
  - 0 : Wide Angle camera
  - ~~1 : Zoom Camera~~ NOT SUPPORTED YET
- VisionCbk : IVision.Stub() that returns:
  - onSuccess : string
  - onError: String

CameraStatus Vision.getStatus(cameraId)

Purpose: give the information whether the camera is started/stopped, Tracking or detecting Motion.

Returns :

A CameraStatus object containing

- wether the camera is started or not (with the isStarted() method)
- wether the Tracking is started or not (with the isTracking() method)
- wether the Motion detection is started or not (with the isDetectingMotion() method)

Params :

- cameraId : the Id of the camera you want to start :
    - 0 : Wide Angle camera
    - ~~1 : Zoom Camera~~ NOT SUPPORTED YET

Object detection

arucoMarkers Vision.detectArucoMarkers()

Purpose: detect the aruco Markers (april Tag) with the camera. The marker must be from the 36h11 dictionary.

Returns :

An arucoMarkers object containing

- The list of x, y of the detected aruco markers. In % (value between [0;1] of the image sides, with the origin at the top-left corner of the image.
- The list of ids of the detected aruco markers

Detections Vision.detectFace(thres)

Purpose: detect the human faces with the camera

Returns :

A Detections object containing

- The list of position of the detected bounding boxes. In % (value between [0;1] of the image sides, with the origin at the top-left corner of the image.
    - Left side horizontal coordinate (with getLeftPos())
    - Right side horizontal coordinate (with getRightPos())
    - Top side vertical coordinate (with getTopPos())
    - Bottom side vertical coordinate (with getBottomPos())
- The list of score (= confidence) of the detections
- The number of detections

Params :

- [Optional] Thres (float) : value between [0 ;1], minimum confidence to detect

DEFAULT VALUE = 0.8

Detections Vision.detectPerson(thres)

Purpose: detect the human with the camera

Returns :

A Detections object containing

- The list of position of the detected bounding boxes. In % (value between [0;1] of the image sides, with the origin at the top-left corner of the image.
    - Left side horizontal coordinate (with getLeftPos())
    - Right side horizontal coordinate (with getRightPos())
    - Top side vertical coordinate (with getTopPos())
    - Bottom side vertical coordinate (with getBottomPos())
- The list of score (= confidence) of the detections
- The number of detections

Params :

- [Optional] Thres (float) : value between [0 ;1], minimum confidence to detect

DEFAULT VALUE = 0.8

Motion detection

The motion detection uses the Farneback optical flow method to detect motion in the image.

As the motion detection analyses the video stream, the motion detection is processed continuously. So the general workflow would be:

- Manually start the motion detection process
- Get the on-demand information of movement, as many times as you like
- Manually stop the motion detection process when no more needed

!!! Not stopping the process will slow down any other computer vision algorithm

void Vision. startMotionDetection ()

Purpose: start the motion detection process

Return :

- void

void Vision. stopMotionDetection ()

Purpose: stop the motion detection process

Returns :

- void

boolean Vision. motionDetect ()

Purpose: detect motion in front of the camera

Returns :

- True : if the optical flow value is > Threshold (10.0 by default)
- False: otherwise

void Vision. setMotionThres (Thres)

Purpose: set the Threshold for motion detection

Params :

- Thres (Float) : the threshold value which defines the motion detection

Returns :

- Void

void Vision. motionDetectWithThres (Thres)

Purpose: detect motion in front of the camera for the specified Threshold

Params :

- Thres (Float) : the threshold value which defines the motion detection

Returns :

- True : if the optical flow value is > Threshold (10.0 by default)
- False: otherwise

motionDetection Vision. getMotionDetection ()

Purpose: get the motion detection

Returns : a motionDetection object which contains

- amplitude (Float) : the amplitude of the highest measured optical flow
- posX (Float) : the position of the highest measured optical flow in the image (origin at the top-left corner). Expressed in %, along the horizontal axis.
- posY (Float) : the position of the highest measured optical flow in the image (origin at the top-left corner). Expressed in %, along the vertical axis.

Color recognition

Colors Vision. ColorDetect ()

Purpose: get the dominant color in the image

Returns : a Colors Enum with:

BLUE, GREEN, YELLOW, ORANGE, RED, PURPLE, WHITE;

Person Tracking

The Person tracking algorithm allow to visually lock on a single person (*target*).

The visual tracking uses the Discriminative Correlation Filter with Channel and Spatial Reliability implementation from OpenCV. (empirically, the best speed/perf. ratio we found, even with a moving camera)

As a tracker uses the previous frame, it works continuously; so the method has to be started (and expicitely stopped)

So the general workflow would be:

- Manually start the tracking process
- Get the on-demand of the tracked target
- Manually stop the tracking process

!!! Not stopping the process will slow down any other computer vision algorithm

void Vision. startVisualTracking(thres, TrackingMode)

Purpose: start the tracking process

Returns :

- Void

Params :

- [OPTIONAL] Thres (Float) : value between [0;1], the threshold value which defines the minimum confidence for the initial human detection
  DEFAULT VALUE=0.8
- [OPTIONAL] TrackingMode (TrackingMode) : TrackingMode.NORMAL or TrackingMode.FAST for a faster but less robust tracking
  DEFAULT VALUE= TrackingMode.NORMAL

void Vision. stopVisualTracking()

Purpose: stop the tracking process

Returns :

- void

Tracking Vision. getTracking()

Purpose: get the tracked target

Returns : a Tracking object which contains

- isTrackingSuccessfull (Boolean) : True if a person is successfully tracked, False if the tracking is lost or nobody is present in the image
- leftPos, rightPos, bottomPos, topPos (int) : The position of tracked bounding box. In % (value between [0;1] of the image sides, with the origin at the top-left corner of the image.

Get the frame from the camera

Bitmap Vision. getGrandAngleFrame(~~VisionCbk~~)

Purpose: get the image frame from the came wide angle ("*Grand angle*") camera

Returns : the camera frame as a Bitmap of 640x480 pixels

Params :

VisionCbk : IVision.Stub() that returns:

- ~~onSuccess : string~~
- ~~onError: String~~

deprecated: use method without argument instead

Bitmap Vision. getCVResultFrame(~~VisionCbk~~)

Purpose: get the image generated from the last executed CV algorithm

Returns : a Bitmap of 640x480 pixels

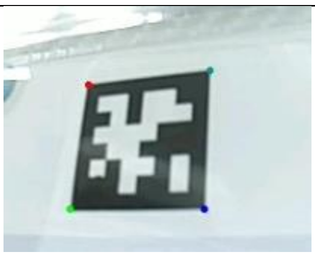
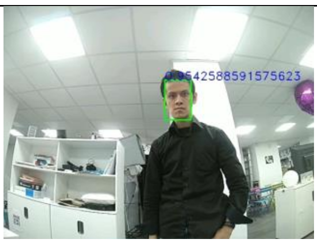
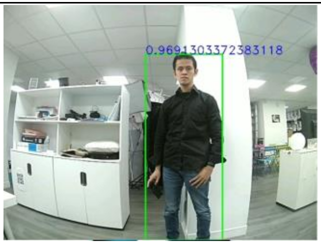
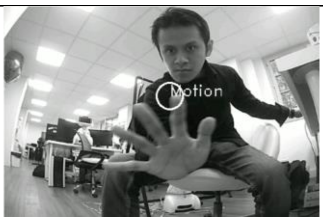
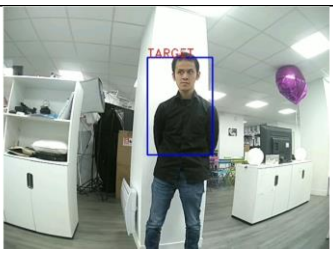
Params :

VisionCbk : IVision.Stub() that returns:

- ~~onSuccess : string~~
- ~~onError: String~~

deprecated: use method without argument instead

Examples of image returned by each CV algorithm:

Aruco Marker detection	
Face Detection	
Human detection	
Motion Detection	
Human Tracking	

6. Behaviour Instructions (BI)

BehaviourInterpreter

This class interpretes and run Behaviour Algorithms which contains sequential or parallel instructions to play robot behaviours.

List of functions in BehaviourInterpreter:

- boolean Run(Context iContext, BehaviourAlgorithm iAlgorithm, OnRunInstructionListener iListener, OnBehaviourAlgorithmListener iAlgorithmEndListener, ImageView iImageView, VideoView iVideoView)

Purpose: Runs the behaviour algorithm given in parameter. This is the method to call if the sequence contains behaviour instruction to display image or videos.

Params:

- **iContext** the context of the activity
- **iAlgorithm** the algorithm to run
- **iListener** will be called each time a behaviour instruction is run
- **iAlgorithmEndListener** will be called when the sequence has ended
- **iImageView** the image view to display images
- **iVideoView** the video view to display videos

return true

```
public boolean Run(Context iContext, BehaviourAlgorithm iAlgorithm,
OnRunInstructionListener iListener, OnBehaviourAlgorithmListener iAlgorith-
mEndListener, ImageView iImageView, VideoView iVideoView)
```

- void Stop()

  Purpose: Stop the algorithm execution.

```
public void Stop()
```

- boolean RunRandom(Context iContext, String iCategory, OnRunInstructionListener iListener, OnBehaviourAlgorithmListener iAlgorithmEndListener, ImageView iImageView, VideoView iVideoView)

!!! :

Purpose: Runs a random standard behaviour from the given category.

Params:

- iContext the context of the activity
- iCategory the category of the bi to choose
- iListener will be called each time a behaviour instruction is run
- iAlgorithmEndListener will be called when the sequence has ended
- iImageView the image view to display images
- iVideoView the video view to display videos

return true if found a bi with the given category, false otherwise

The categories usable are the following:

Angry, Awake, BadAnswer, BlinkDouble, BlinkLeft, BlinkRight, CenterHead, CenterHeart, Cliff-Back, CliffFront, CliffLift, Congratulations, Dance, Defeat, Demo, DemoShort, DetectSound, Doc-torCall, Doubtful, FastHeartBeat, FoundSomeone, GoodAnswer, Growling, Grumpy, Happy, Idle, Idle_ANGRY, Idle_HAPPY, Idle_SAD, Idle_TIRED, InactivityDetected, Joke, LeftHead, LeftShoulder, Listening, Love, LowHeartBeat, Neutral, OveractivityDetected, RightHead, RightShoulder, Sad, Scared, Sick, Sleep, Smile, Surprised, Suspicious, Tease, Thinking, Tired, TofBack, TofFront, TrackingEnd, TrackingStart, Victory, WakeUp, WatchNotWorn, What, Whistle, and Yawn

- registerOnRunInstructionListener(OnRunInstructionListener iLis-tener)

  Purpose: register to a class that implement OnRunInstructionListener
  Param:
- iListener : class that implement OnRunInstructionListener

```
public void registerOnRunInstructionListener(OnRunInstructionListener iLis-
tener)
```

- BehaviourAlgorithmStorage Deserialize(Context iContext, String iFile)

Purpose: Deserialize an xml file that contains a BehaviourAlgorithmStorage

Params:

- iContext the context of the activity
- iFile the name of xml file that contains the algorithm (the xml must be in the application files folder).
  Return the BehaviourAlgorithmStorage that contains the algorithm

```
public static BehaviourAlgorithmStorage Deserialize(Context iContext,
String iFile)
```

- BehaviourAlgorithmStorage Deserialize(Context iContext, File iFile)

Purpose: Deserialize an xml file that contains a BehaviourAlgorithmStorage

Params:

- iContext the context of the activity
- iFile the xml file that contains the algorithm (the xml must be in the application files folder). Return the BehaviourAlgorithmStorage that contains the algorithm

```
public static BehaviourAlgorithmStorage Deserialize(Context iContext, File
iFile)
```

- void Serialize(Context iContext, BehaviourAlgorithmStorage iStorage, File iFile)

  Purpose : Serialize a BehaviourAlgorithmStorage into an xml
  Params:
- iContext the context of the activity
- iStorage the object to serialize
- iFile the file that will contains the object

```
public static void Serialize(Context iContext, BehaviourAlgorithmStorage
iStorage, File iFile)
```

OnRunInstructionListener

This interface is used to get the bi that is currently running by the interpreter

- void OnRunInstruction(ABehaviourInstruction instruction)
  Purpose: Callback called each time a new behaviour instruction is runnning.
  Params:
- instruction the current behaviour instruction currently runnning.

```
void OnRunInstruction(ABehaviourInstruction instruction);
```

OnBehaviourAlgorithmListener

This interface is used to know at what moment the sequence playing has stopped (either by using the stop method or if it has ended).

- void OnBehaviourAlgorithm(boolean hasAborted)
  Purpose: Will be called when the algorithm execution has ended
  Params:
- hasAborted is true if the sequence has been aborted (by calling the stop method from BehaviourInterpreter). If it has ended normally it's false.

```
void OnBehaviourAlgorithm(boolean hasAborted);
```

Example to read a BI:

Declare A behaviour interpreter in your activity class

```
private BehaviourInterpreter interpreter;
```

Then initialize it in OnSdkReady

```
interpreter = new BehaviourInterpreter();
```

Before using the following function you can implements the OnRunInstructionListener and BehaviourAlgorithmListener in your activity class

```
public class MainActivity extends BuddyActivity implements OnRunInstruc-
tionListener, OnBehaviourAlgorithmListener
```

This function read a BI and executes it:

```
private void readBI(String biName) {


    ImageView imageView =findViewById(R.id.imageView);

    VideoView videoView=findViewById(R.id.videoView);

    String docPath = Environment.getExternalStoragePublicDirectory(Environ-
ment.DIRECTORY_DOWNLOADS).toString();

    String fileName = docPath + "/" + biName;

    File source = new File(fileName);


    try {

        //BehaviourAlgorithmStorage storage = serializer.read(Behaviour-
AlgorithmStorage.class, source);

        BehaviourAlgorithmStorage storage = interpreter.Deserialize(this,
source);

        if(storage==null) {


            Log.e(TAG, "onReadBI storage null");

        }

        else {

            final boolean run = interpreter.Run(this, storage.getAlgo-
rithm(), this, this, imageView, videoView);

            biPlaying=true;

        }

    } catch (Exception e) {

        e.printStackTrace();

    }

}
```

The xml must be in the download folder in Android

You must have a videoview and imageView in your layout application in order to use the previous function.

7. User interface (UI)

- void *UI*.setCloseWidgetVisibility(FloatingWidgetVisibility iVisibility);

Purpose: set the visibility of the close Button in your app. This Button is managed by Buddy-Core and is always visible by default. However you can change the visibility of the button in your app using this method.

Params:

- iVisibility (FloatingWidgetVisibility enum) :
  - FloatingWidgetVisibility.NEVER : completely hide the button
  - FloatingWidgetVisibility.ALWAYS : always show the button
  - FloatingWidgetVisibility.ON_TOUCH : only shows the button after a touch of the caress sensors or the screen[2]

- void *UI*.setMenuWidgetVisibility(FloatingWidgetVisibility iVisibility);

Purpose: set the visibility of the BuddycoreMenu Button in your app. This button is managed by BuddyCore and is always visible by default. However you can change the visibility of the button in your app using this method.

Params:

- iVisibility (FloatingWidgetVisibility enum) :
  - FloatingWidgetVisibility.NEVER : completely hide the button
  - FloatingWidgetVisibility.ALWAYS : always show the button

[2] The activity you touch must have called setViewFace

    - FloatingWidgetVisibility.ON_TOUCH : only shows the button after a touch of the caress sensors or the screen[3]

- void UI. startPinLock(String iCode, String iTitle, Consumer<Boolean> iCallback);

Purpose: Lock screen until user enters PIN code or closes dialog. Can be also unlocked with dev code (if set) and master code.

Params:

- ICode (String) : PIN code. Max 6 characters, numbers only.
- iTitle (String): Optional title to show.
- iCallback (Consumer<Boolean>): True if code correct, false if code incorrect or closed.

- void UI. askToQuitApp(String iMessage, int iAutoHideTime);

Purpose: Show a popup asking user whether to quit the application. It is automatically called with default parameters when top-right "Close" button is tapped.

Params:

- iMessage (String) : Message to show. If empty, it is set to default "Are you sure you want to close this app?"
- iAutoHideTime (String) : Time in seconds. Automatically close app if no user response. Default is 0 (no autoclose).

- void UI. setQuitButtonDialogMessage(String iMessage);

Purpose: Change message that is shown when close button is tapped.

Params:

- iMessage (String) : Message to show. If empty, it is set to default "Are you sure you want to close this app?"

: in any case, the BuddyCore Menu button and the close app button appear after maintaining the contact on both the head and heart caress sensors simultaneously during 10s.

[3]The activity you touch must have called setViewFace

8. Companion

void Companion.raiseEvent(String eventName, Map<String, String> eventParameters)

Purpose: Raise a Companion event. This event can indirectly trigger a mission that is already listening to this event.

Parameters:

eventName (String): Name of the event

(optional) eventParameters (Map<String, String>): Parameters of the event. If the associated mission calls a runActivity, the parameters will be forwarded in the extras of the launched activity.

8 - TUTORIALS

1) Make the robot move (wheels)

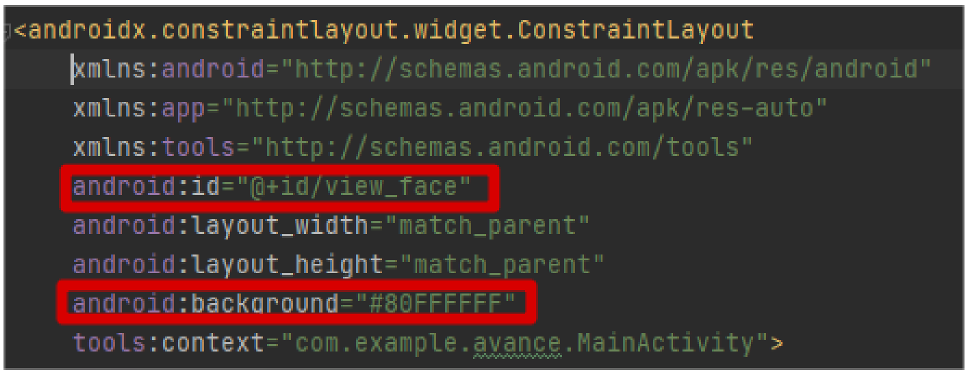
In the following we present a tutorial to make the robot go forward

- Layout XML file:

The aim of the app is to make Buddy go forwards.

First, define the layout.

Step 1: At the start of your code it's important to add the red squared lines for the interface.

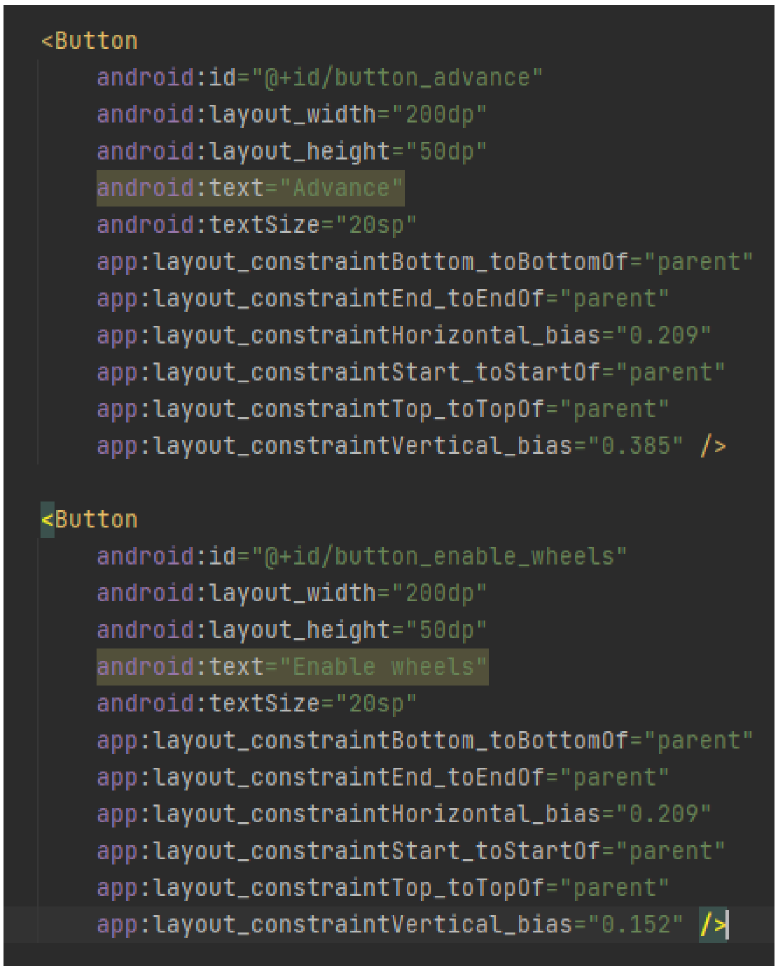


```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/view_face"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#80FFFFFF"
    tools:context="com.example.avance.MainActivity">
```

Step2: Add two buttons to control the robot.

The first will be used to start the wheels, it will enable the wheels with the *enableWheels()* function. Wheels must be enabled before any mouvement to be successful.

The second button will command the deplacement of the robot with the *moveBuddy()* function.



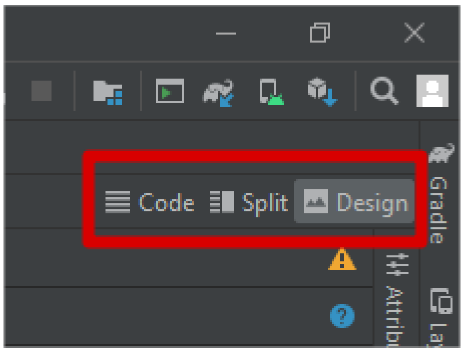
```
<Button
    android:id="@+id/button_advance"
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:text="Advance"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.209"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.385" />

<Button
    android:id="@+id/button_enable_wheels"
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:text="Enable wheels"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.209"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.152" />
```

The "**advance**" button is used to, once it's clicked on, make Buddy move.

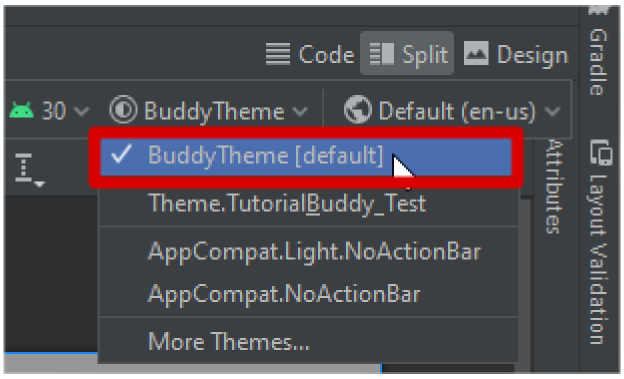
The "**Enable wheels**" button is used to, once it's clicked on, enable the wheels motor.

Important ! Give each of the buttons an Id, it will be important in the programmation of the java file.

Step3: Finally, on the top right of your screen, go in the Design part.



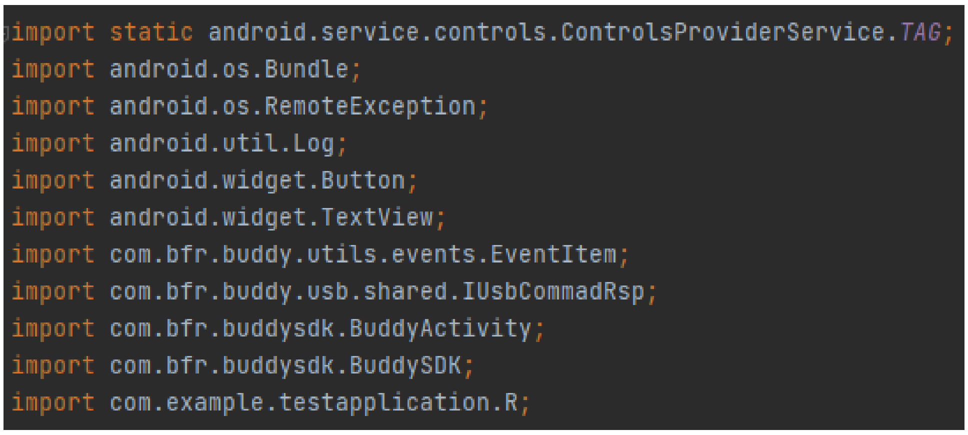
choose "**BuddyTheme [default]**"



- MainActivity file:

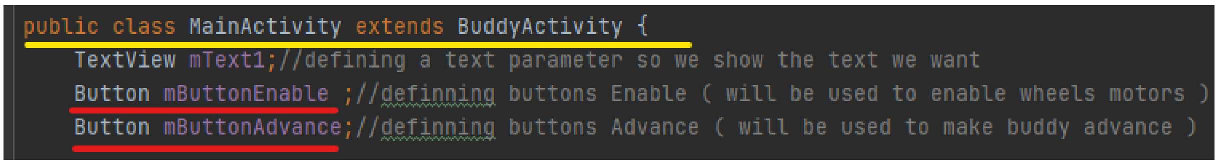
Step 1: Go in the MainActivity Java file and start by importing the needed functions. Here are the ones needed to move Buddy.

: Anytime you miss an import/reference in your java code (appears in red), right click on it and press ALT + ENTER. The auto completion from Android Studio will add it for you!



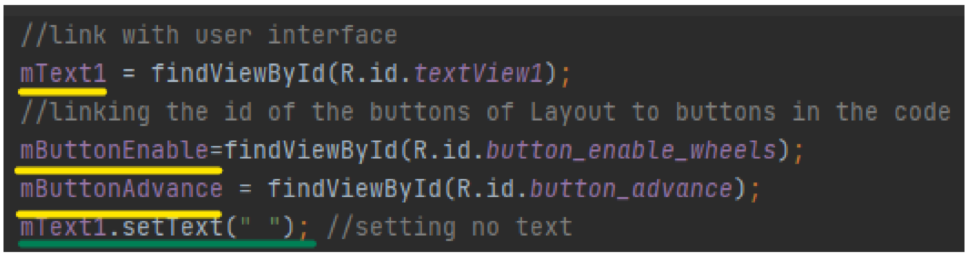
```java
import static android.service.controls.ControlsProviderService.TAG;
import android.os.Bundle;
import android.os.RemoteException;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;
import com.bfr.buddy.utils.events.EventItem;
import com.bfr.buddy.usb.shared.IUsbCommadRsp;
import com.bfr.buddysdk.BuddyActivity;
import com.bfr.buddysdk.BuddySDK;
import com.example.testapplication.R;
```

Step 2: In the **public class MainActivity**, define 2 buttons for each button on the layout file



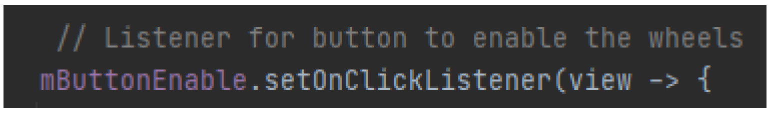
```java
public class MainActivity extends BuddyActivity {
    TextView mText1;//defining a text parameter so we show the text we want
    Button mButtonEnable ;//definning buttons Enable ( will be used to enable wheels motors )
    Button mButtonAdvance;//definning buttons Advance ( will be used to make buddy advance )
```

In the **onCreate** location, link the two buttons with the two created in the layout file.



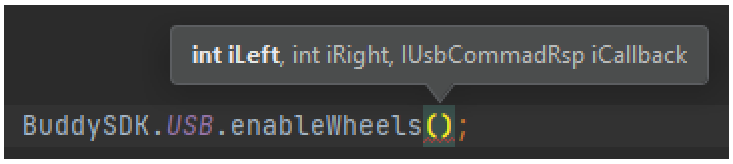
```
//link with user interface
mText1 = findViewById(R.id.textView1);
//linking the id of the buttons of Layout to buttons in the code
mButtonEnable=findViewById(R.id.button_enable_wheels);
mButtonAdvance = findViewById(R.id.button_advance);
mText1.setText(" "); //setting no text
```

Step 3: In onCreate, set the button enable_wheels OnClickListner, this will execute the command if the button get clicked



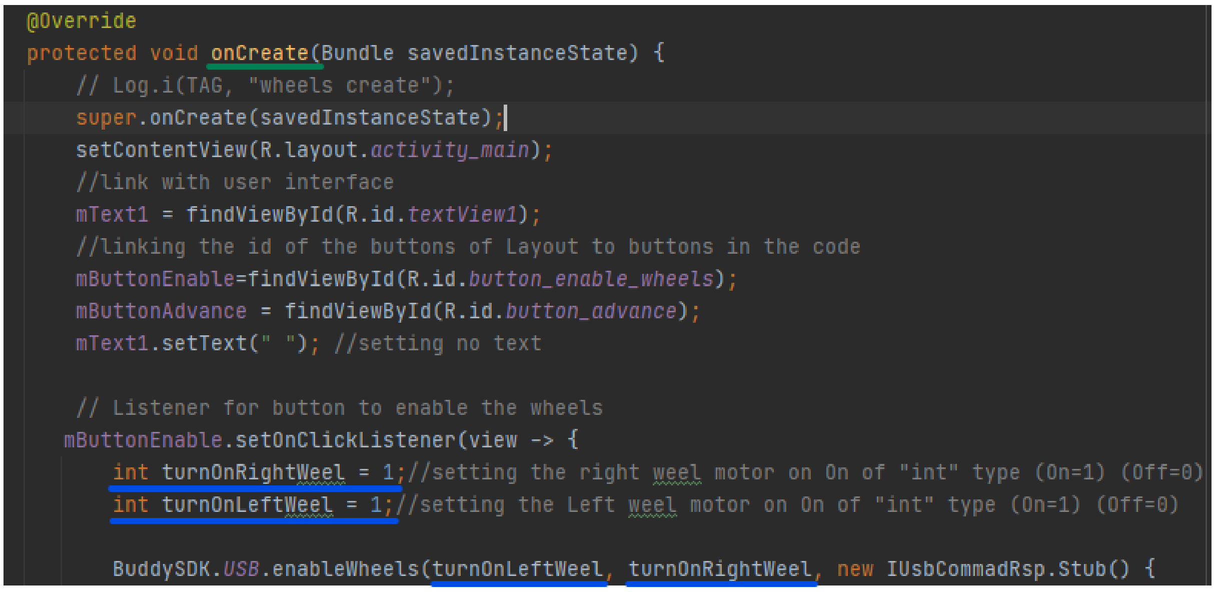
```
  // Listener for button to enable the wheels
 mButtonEnable.setOnClickListener(view -> {
```

The function linked to this OnClickListener is *enableWheels()* here.



```
int iLeft, int iRight, IUsbCommadRsp iCallback
BuddySDK.USB.enableWheels();
```

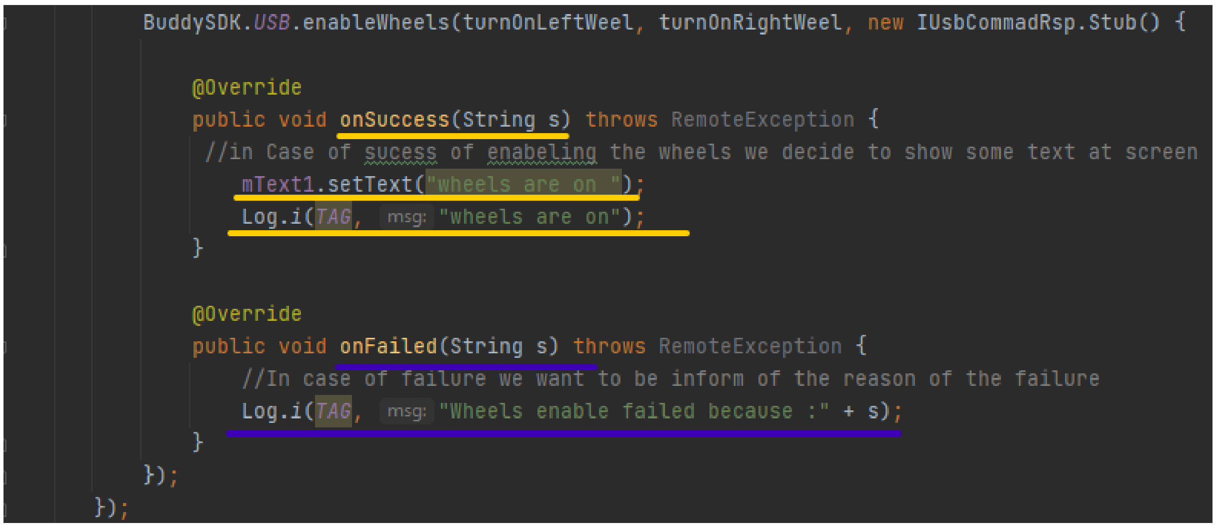
Step 4: The motors wheels have to be enabled, set the parameters of the enableWheels function turnOnRight and turnOnLeft to 1.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // Log.i(TAG, "wheels create");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //link with user interface
    mText1 = findViewById(R.id.textView1);
    //linking the id of the buttons of Layout to buttons in the code
    mButtonEnable=findViewById(R.id.button_enable_wheels);
    mButtonAdvance = findViewById(R.id.button_advance);
    mText1.setText(" "); //setting no text

    // Listener for button to enable the wheels
   mButtonEnable.setOnClickListener(view -> {
       int turnOnRightWeel = 1;//setting the right weel motor on On of "int" type (On=1) (Off=0)
       int turnOnLeftWeel = 1;//setting the Left weel motor on On of "int" type (On=1) (Off=0)

       BuddySDK.USB.enableWheels(turnOnLeftWeel, turnOnRightWeel, new IUsbCommadRsp.Stub() {
```

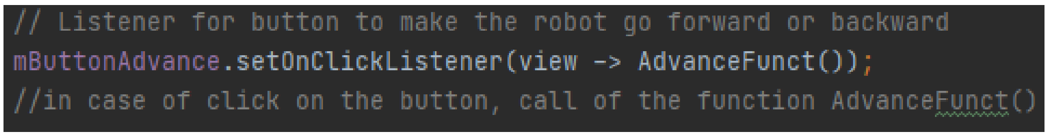
Step 5: Use the *IUsbCommadRsp.Stub()* to get the success or failed callback with the "s" string.

It's also possible to add specific log for those callbacks.

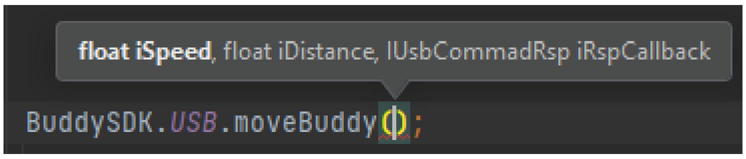

```
        BuddySDK.USB.enableWheels(turnOnLeftWeel, turnOnRightWeel, new IUsbCommadRsp.Stub() {

            @Override
            public void onSuccess(String s) throws RemoteException {
             //in Case of sucess of enabeling the wheels we decide to show some text at screen
               mText1.setText("wheels are on ");
               Log.i(TAG,  msg: "wheels are on");
            }

            @Override
            public void onFailed(String s) throws RemoteException {
               //In case of failure we want to be inform of the reason of the failure
               Log.i(TAG,  msg: "Wheels enable failed because :" + s);
            }
        });
    });
```

Now the motor of each wheel is set on, the robot can move with different functions

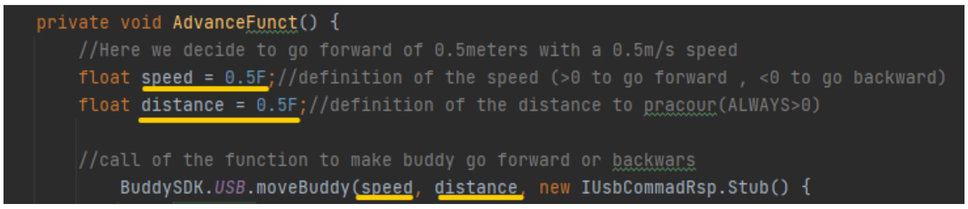
Step 6: In onCreate, set the button mButtonAdvance OnClickListner, this will execute the AdvanceFunction if the button get clicked


```
// Listener for button to make the robot go forward or backward
mButtonAdvance.setOnClickListener(view -> AdvanceFunct());
//in case of click on the button, call of the function AdvanceFunct()
```

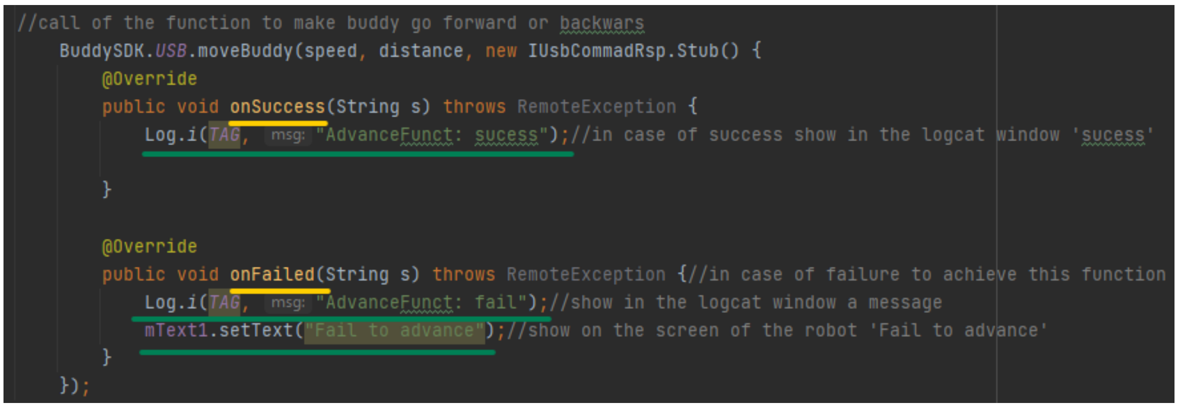
This button will call *moveBuddy()*.


```
float iSpeed, float iDistance, IUsbCommadRsp iRspCallback
BuddySDK.USB.moveBuddy();
```

Step 7: Define the speed and distance settings for the *moveBuddy()* function in the *AdvanceFunct()*


```
private void AdvanceFunct() {
    //Here we decide to go forward of 0.5meters with a 0.5m/s speed
    float speed = 0.5F;//definition of the speed (>0 to go forward , <0 to go backward)
    float distance = 0.5F;//definition of the distance to pracour(ALWAYS>0)

    //call of the function to make buddy go forward or backwars
        BuddySDK.USB.moveBuddy(speed, distance, new IUsbCommadRsp.Stub() {
```

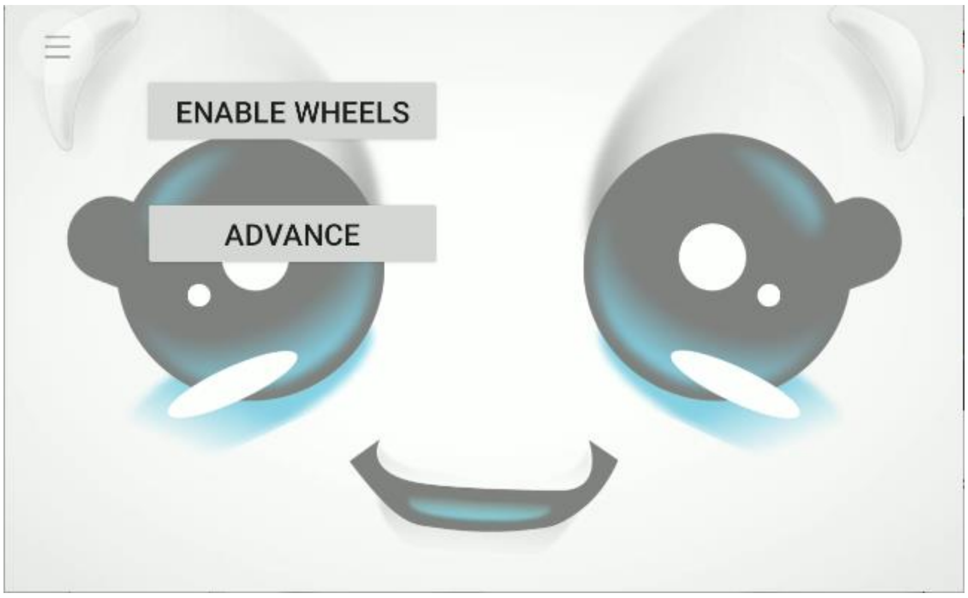
Step 8: The callbacks function is set by IUsbCommadRsp.Stub() of this moveBuddy function. Once again it is possible to customize the callback in the yellow underline zone to ensure of success or failure of each part.



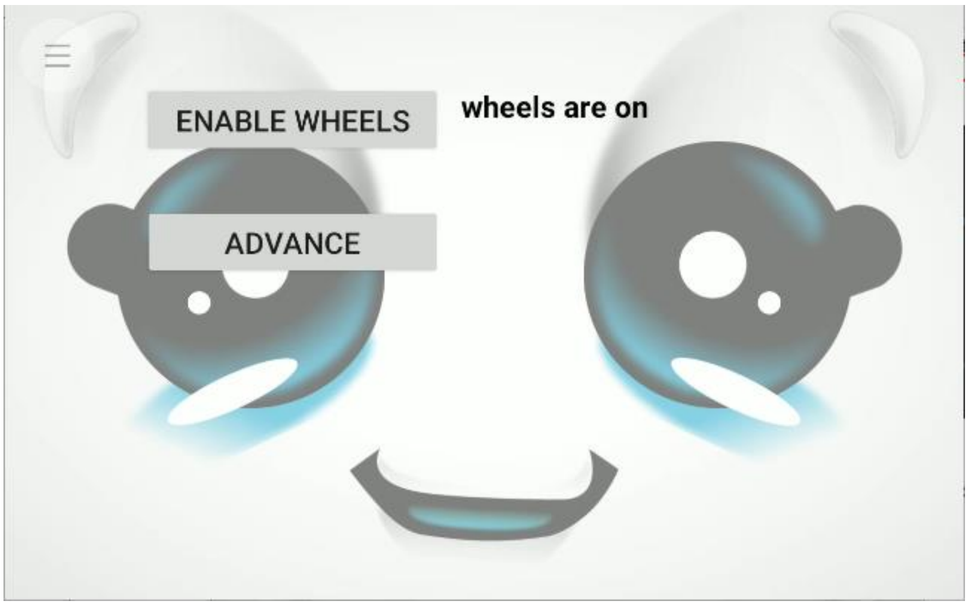
```
//call of the function to make buddy go forward or backwars
    BuddySDK.USB.moveBuddy(speed, distance, new IUsbCommadRsp.Stub() {
        @Override
        public void onSuccess(String s) throws RemoteException {
            Log.i(TAG,  msg: "AdvanceFunct: sucess");//in case of success show in the logcat window 'sucess'

        }

        @Override
        public void onFailed(String s) throws RemoteException {//in case of failure to achieve this function
            Log.i(TAG,  msg: "AdvanceFunct: fail");//show in the logcat window a message
            mText1.setText("Fail to advance");//show on the screen of the robot 'Fail to advance'
        }
    });
```

2) Running the App

Step 1: Launch the app and this display should appear:



Step 2: Click on Enbale Weels and wait for the callback.



Step 3: Click on **Advance** and Buddy should move forwards at 0.5m/s on 50cm.

2) Make the robot move (HEAD)

In the following, we present an example of an application to move the head of buddy with the SDK.

- Layout XML file :

Step 1*: Create new buttons to do the different moves for head.*

Here is the button for Buddy to do a "yes move".

```
<Button
    android:id="@+id/button_yes"
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:text="MOVE YES"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.047"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.08" />
```

Here is the switch to enable the motor for "yes move"

```
<Switch
    android:id="@+id/Enable_yes"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Enable no"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Step 2: Create two EditText to enter the value of angle you want

Here is the EditText for the angle we have choosen for Buddy to do a "yes move"

```
<EditText
    android:id="@+id/angle_yes"
    android:layout_width="244dp"
    android:layout_height="139dp"
    android:ems="10"
    android:gravity="center"
    android:hint="Specify the angle of Yes"
    android:inputType="numberSigned"
    android:scaleType="fitCenter"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.05"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.667" />
```

- MainActivity file :

Step1: Go to MainActivity and initialize the button to do a "yes move" and a switch to enable "yes move" motor

Button :

```
findViewById(R.id.button_yes).setOnClickListener(v -> onButtonYes());//The button
allowing Buddy to do a "yes" move
```

Switch :

```
private Switch Enable_switch_no; //switch to enable motor for "no" move
```

```
Enable_switch_no = findViewById(R.id.Enable_no); //Linking between xml switch and
Enable_switch_no variable
```

We recommend to do the initialization at the beginning, for instance, in the onCreate() callback of your application.

Step 2: Create the function to enable the motor.

To save power, the motors are disabled by default. Enable the motors when the switch is checked. In our case we focus on the motor to do "yes move".

```
//Switch to enable or disable the motor
Enable_switch_yes.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The motor for "yes" move is enable
            BuddySDK.USB.enableYesMove(1, new IUsbCommadRsp.Stub() {
                @Override
                //if the motor succeeded to be enabled,we display motor is enabled
                public void onSuccess(String success) throws RemoteException {
                    Log.i("Motor Yes", "Yes motor Enabled");
                }

                @Override
                //if the motor did not succeed to be enabled,we display motor
failed to be enabled
                public void onFailed(String error) throws RemoteException {
                    Log.i("Motor Yes", "Yes motor Enabled Failed");
                }
            });
        } else    // The motor for "yes" move is enable
        {
            BuddySDK.USB.enableYesMove(0, new IUsbCommadRsp.Stub() {
                @Override
                //if the motor succeeded to be disabled,we display motor is
disabled
                public void onSuccess(String success) throws RemoteException {
                    Log.i("Motor Yes", "Yes motor Disabled");
                }

                @Override
                //if the motor did not succeed to be disabled,we display motor
failed to be disabled
                public void onFailed(String error) throws RemoteException {
                    Log.i("Motor Yes", "Yes motor Disable Failed");
                }
            });
        } // end if checked
    } // end Onchecked callback
});
```

If you want to enable the motor for Buddy to say yes, the following line will be useful.

This line allows you to enter the arguments.

```
       BuddySDK.USB.enableYesMove(State,Callback ()
```

These are the arguments :

- State : motor enabled or not (respectively 1 or 0)
- Callback for return, "OK" if success and "NOK" if fail

Step 3: Create the Editable text where you will select the angle value you want

```
EditText angle_Yes; //Editable text to insert an angle value for "Yes" move
```

Step 4: Setup the methods which allow to do a "yes move".

When we click on the button which allow Buddy to do a "yes move", this methods is launched.

```
private void onButtonYes() {
       //Buddy function to do a "yes move"
BuddySDK.USB.buddySayYes(10,Integer.parseInt(String.valueOf(angle_Yes.getText())),
new IUsbCommadRsp.Stub()   {
        @Override
        //If buddySayYes succeed to finish his "yes" move,success take the
value"YES_MOVE_FINISHED"
        public void onSuccess(String success)
        {
               //When success takes the value "YES_MOVE_FINISHED",buddy  will
bring his head back
            if (success.equals("YES_MOVE_FINISHED"))
            {
                BuddySDK.USB.buddySayYes(10, -
Integer.parseInt(String.valueOf(angle_Yes.getText())), null);
            }
        }

        @Override
        //if the function did not succeed,nothing is happening
        public void onFailed(String error) {
        }
    });
}
```

If you want Buddy to say **yes**, you will need to use this line.

This line allows you to enter the **arguments**.

```
   BuddySDK.USB.buddySayYes(Speed,Angle,Callback()
```

These are the **arguments** :

- **Speed** : angular speed in **°/s** between –49.2 and 49.2
- **Angle** : angle in **°** between –45and 45**.**
- **Callback** for return, "OK" if launched, "YES_MOVE_FINISHED" if move finished and "NOK" if failed

```
//Buddy function to do a "yes move"
BuddySDK.USB.buddySayYes(10,Integer.parseInt(String.valueOf(angle_Yes.getText())),
new IUsbCommadRsp.Stub()
```

This line allows Buddy to execute something if the **method** worked.

```
if (success.equals("YES_MOVE_FINISHED"))
```

These lines allow Buddy to execute another **BuddySayYes** methods for Buddy to do a "**yes move**".

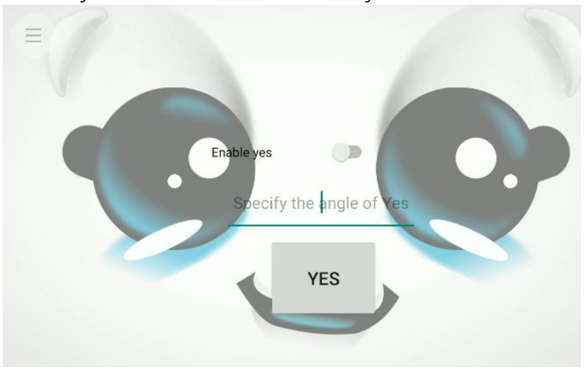
```
//If buddySayYes succeed to finish his "yes" move,success take the
value"YES_MOVE_FINISHED"
public void onSuccess(String success)
{
       //When success takes the value "YES_MOVE_FINISHED",buddy  will bring his
head back
    if (success.equals("YES_MOVE_FINISHED"))
    {

    }
}
```

This line is the callback if the **method** did not execute well

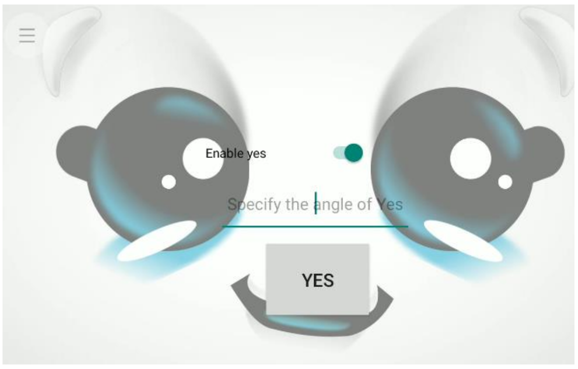
```
    //if the function did not succeed,nothing is happening
    public void onFailed(String error) {
    }
});
```

2) Running the App

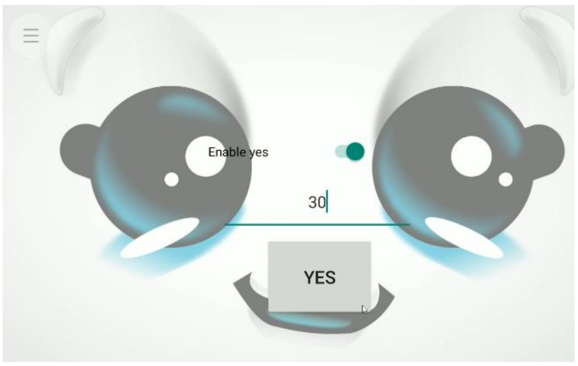
When you launch the method you can can observe a window with the face of Buddy.



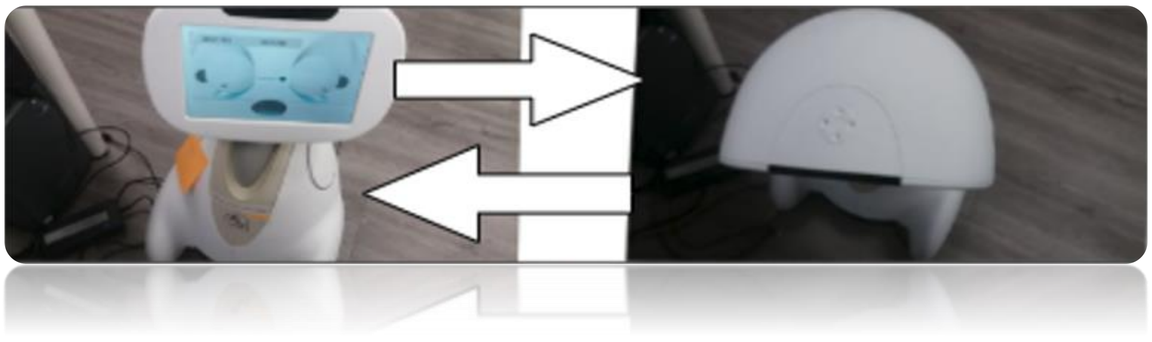
Check the switch to do a "yes move"



Enter the value of angle you want



This move is executed for BuddySayYes



APPENDIX :

1 - Vocon grammars content

The content of the Cerence Grammars (predefined sentences that Buddy can understand through its STT functionality) are in annex.

2 - BNF COMPILATION

Backus-Naur form (BNF) is a formal notation for encoding speech to text grammars. (cf documentation in Bfn_compilation_windows_tools.zip Cerence_doc/vocon_grammar_formalisms.html)

A bfn is usaly stored inside a text file. As you can see in the API the functions Speech.createCerenceTask do not take a .bnf file as input but a .fcf. A .fcf is a .bnf compiled. So in order to use the Speech.createCerenceTask functions you need to compile your .bnf. To do that you need to use a windows executable.

Here are the steps to do:

1. Open the archive named Bfn_compilation_windows_tools.zip

2. Go in Compile_bnf_in_<language>/ folder depending on which langage you want to compile.

3. Edit the file grmcpl_samples/audio.bnf according to what you want to listen.

4. Open « Git bash » (for exemple) on the root folder of a language (Compile_bnf_in_<language>/) and type:

For English folder:

```
./grmcpl.exe  --modelFilepath=acmod6_9000_enu_gen_car_f16_v2_0_0.dat -p sample.txt -C grmcpl_samples/results/audio.fcf
```

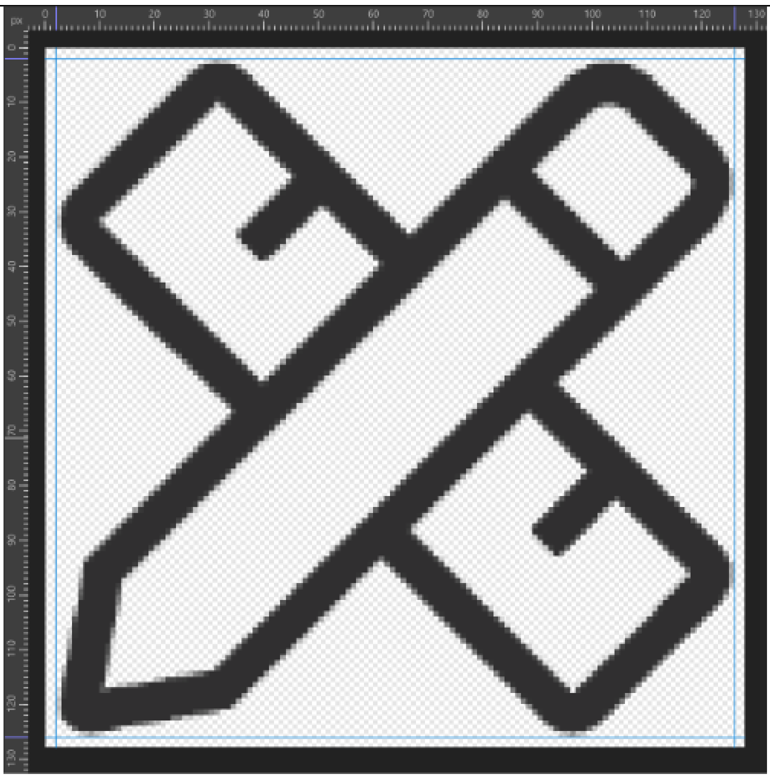
For French folder:

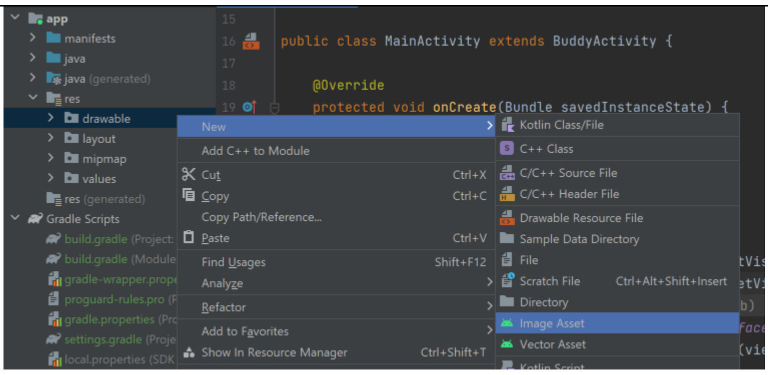
[redacted]

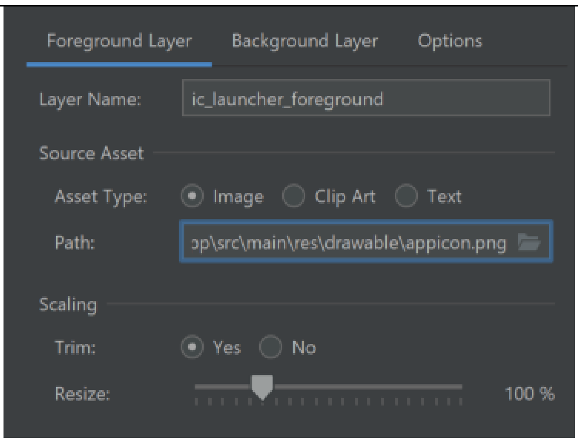
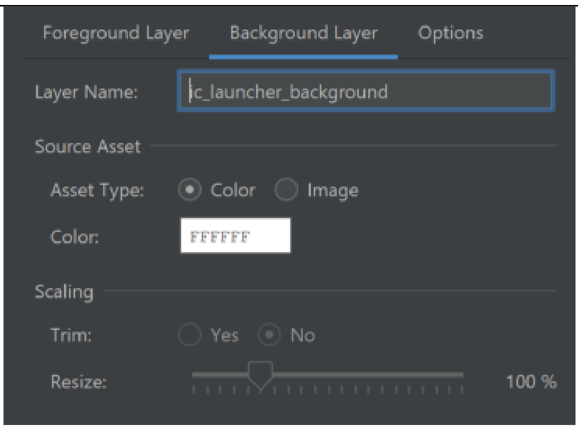
For more details about gmcpl.exe see Cerence_doc/tools/grmcpl/grmcpl.html

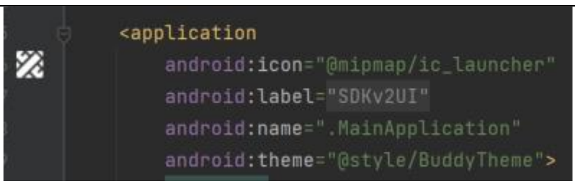
5. Get the generated .fcf in grmcpl_samples/results/audio.fcf

3 - APP ICON

In your image editor, prepare an image file: - 128x128 px - 72 dpi - Transparent background - Black lines - Outline only - Leave around 2 pixel margins just to be sure image is not cut on the edge Save it to \app\src\main\res\drawable in your app's directory as appicon.png	

In Android studio, go to app > res > drawable and click it with right mouse button
 	
Select New > Image Asset
	
	`public class MainActivity extends BuddyActivity {`
	`@Override`
	`protected void onCreate(Bundle savedInstanceState) {`
	 app
	manifests
	java
	java (generated)
	res
	drawable
	layout
	mipmap
	values
	res (generated)
	Gradle Scripts
	build.gradle (Project:
	build.gradle (Module
	gradle-wrapper.prope
	proguard-rules.pro (P
	gradle.properties (Pro
	settings.gradle (Proje
	local.properties (SDK
	New
	Add C++ to Module
	Cut Ctrl+X
	Copy Ctrl+C
	Copy Path/Reference...
	Paste Ctrl+V
	Find Usages Shift+F12
	Analyze
	Refactor
	Add to Favorites
	Show In Resource Manager Ctrl+Shift+T
	Kotlin Class/File
	C++ Class
	C/C++ Source File
	C/C++ Header File
	Drawable Resource File
	Sample Data Directory
	File
	Scratch File Ctrl+Alt+Shift+Insert
	Directory
	Image Asset
	Vector Asset
	Kotlin Script

Set Foreground layer settings : 1. Select your image under Source Asset Path 2. Trim to Yes, Resize 100%	 Foreground Layer · Background Layer · Options Layer Name: ic_launcher_foreground Source Asset Asset Type: [✓ Image] Clip Art Text Path: ɔp\src\main\res\drawable\appicon.png Scaling Trim: [✓ Yes] No Resize: 100 %
Set white color as backgound	 Foreground Layer · Background Layer · Options Layer Name: ic_launcher_background Source Asset Asset Type: [✓ Color] Image Color: FFFFFF Scaling Trim: Yes [✓ No] Resize: 100 %

Press **Next**, then press **Finish**
 	
If saving file shows errors, delete ***mipmap*** *folder and repeat process*
 	
Your manifest file should contain **android:icon** like shown in example
	
	`<application`
	`    android:icon="@mipmap/ic_launcher"`
	`    android:label="SDKv2UI"`
	`    android:name=".MainApplication"`
	`    android:theme="@style/BuddyTheme">`

Correspondance Mood <> Led colors

Angry	#cc0000
Grumpy	#96257c
Happy	#ffc700
Listening	#53b200
Neutral	#00d3d0
Sad	#ff0ab1
Scared	#ccc930
Sick	#628c00
Surprised	#cccc00
Thinking	#35dd40
Tired	#474c20
Love	#aa5a63

List of parameters in applications.json (non-exhaustive)

- *« Package »*
  Application package name
- *« Name »*
  Application name override. If not empty, shows under the icon instead of the real name.
  If neither exist, will show package name.
- « ShowInMenu »
  Whether to show application on Apps menu tab
- *« Autostart »*
  Whether to launch application immediately when Core finishes its starting sequence
- *« ShowMenuButton »*
  Whether to show or hide Menu floating widget.
- *« ShowCloseButton »*
  Whether to show or hide Close floating widget.
- *« AskToQuit »*
  Whether to show close dialog when clicked on close button.
- *« ShowPopups »*
  Whether to show dialog when clicked on close button.