BUDDY SDK

User Guide v 2.4

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
07-11-2022	V2.3.11	Kevin HOANG		No singleInstance
07-02-2023	V2.3.12	Kevin HOANG		Corrections Tutorial
17-03-2023	V2.3.13	Mickael Cocquempot		Launch Companion's mission from SDK
04-04-2023	V.2.3.14	Mickael Cocquempot		Various fixes
06-04-2023	V.2.3.15	Kevin HOANG		Various fixes
11-04-2023	V.2.3.16	Mickael Cocquempot		Various fixes
26-04-2023	V.2.3.17	Mickael Cocquempot		Add informations in the tutorial
01-06-2023	V.2.3.18	Mickael Cocquempot		Change some part of the tutorial

12-06-2023	v.2.3.19	Walid ABDERRAHMANI		Change bi part
16-06-2023	v.2.3.20	Mickael Cocquempot		Add documentation on companion's part
12-07-2023	v.2.3.21	Walid ABDERRAHMANI		Corrected Task name in bi part
06-10-2023	V.2.3.22	Mickael Cocquempot		Add documentation about Realign task
10-11-2023	V.2.3.23	Kevin HOANG		Actuators workflow diagram
26-12-2023	V.2.3.24	Ilya SEMISALOV		Notifications
15-03-2024	V2.4.0	Kevin HOANG		SDK v2.4

1 -TABLE DES MATIERES

1 - PREREQUISITES :

- Basic knowledge on how to program an Android App
- Android studio > 4.2
- Android SDK >=30
- 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

It is also recommended to know the basic adb commands like “pull, push, install, shell,...” to fully explore the possibility of your Buddy

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**.

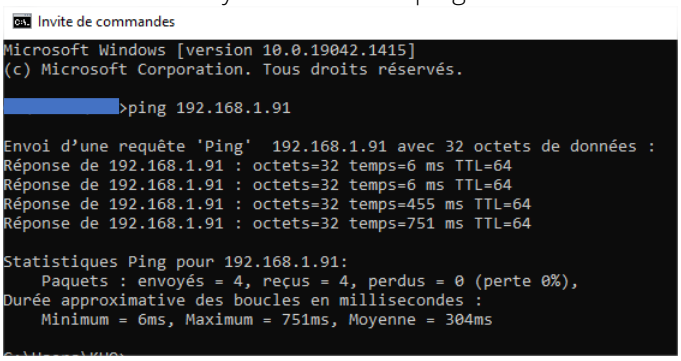
Highly recommended : adb is also provided within **Screencopy,** a useful tool to monitor/control your device from your computer. Go to https://github.com/Genymobile/scrcpy, and install it following the instructions in the “Get the app “ section.

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:

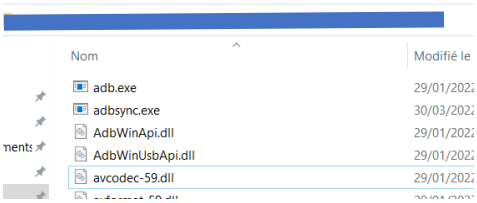
   - Click on the Connection icon and get the IP address of your robot in the Wifi section

2bis. Check if you can at least ping it

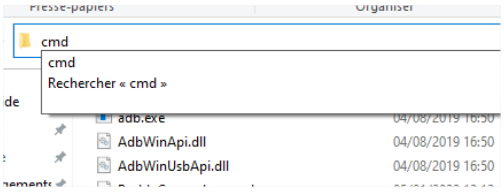


On Windows:

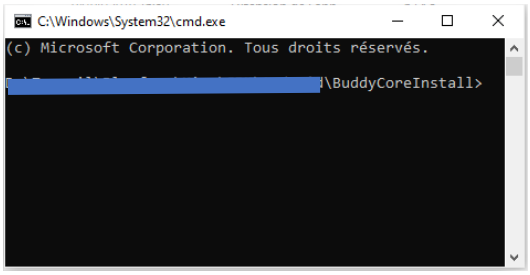
3. Go to a folder containing "adb.exe"



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

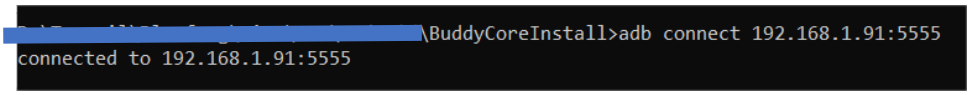


5. A command invite should appear like this :



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...)



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.

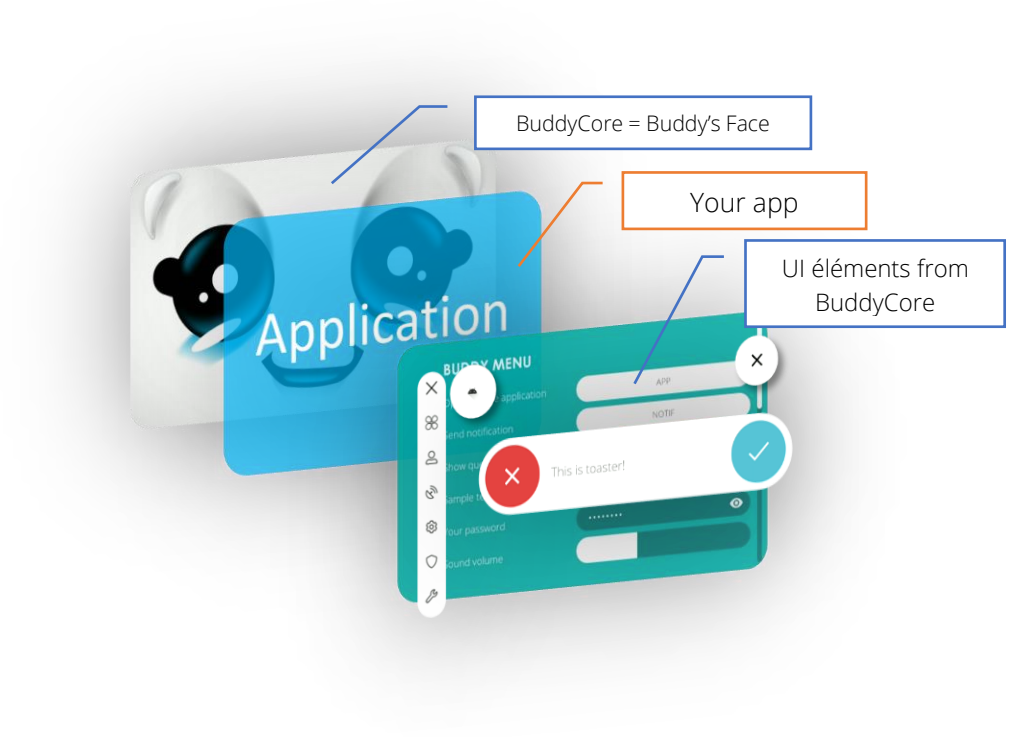
For your app to work properly, BuddyCore must always be launched in the background, so that your app will appear on top of it. That's why, BuddyCore should be the default Launcher on your device.

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.



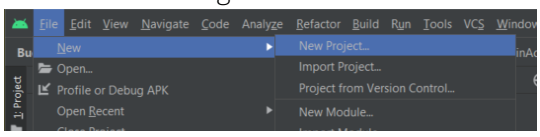
The following explains how to use the BFR SDK within Android Studio from scratch. If you already have an Android project, you can skip directly to the section "Import the SDK module in your project".

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

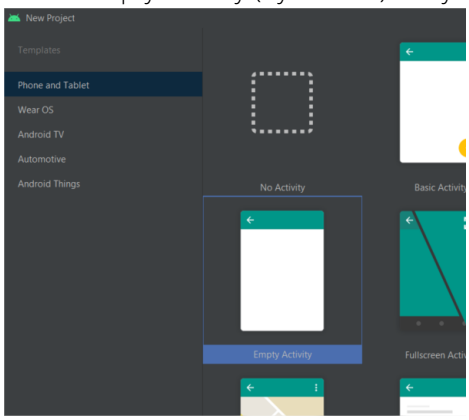
Also, a ready-to-use code template is also provided with the examples.

2) Create an empty Android App

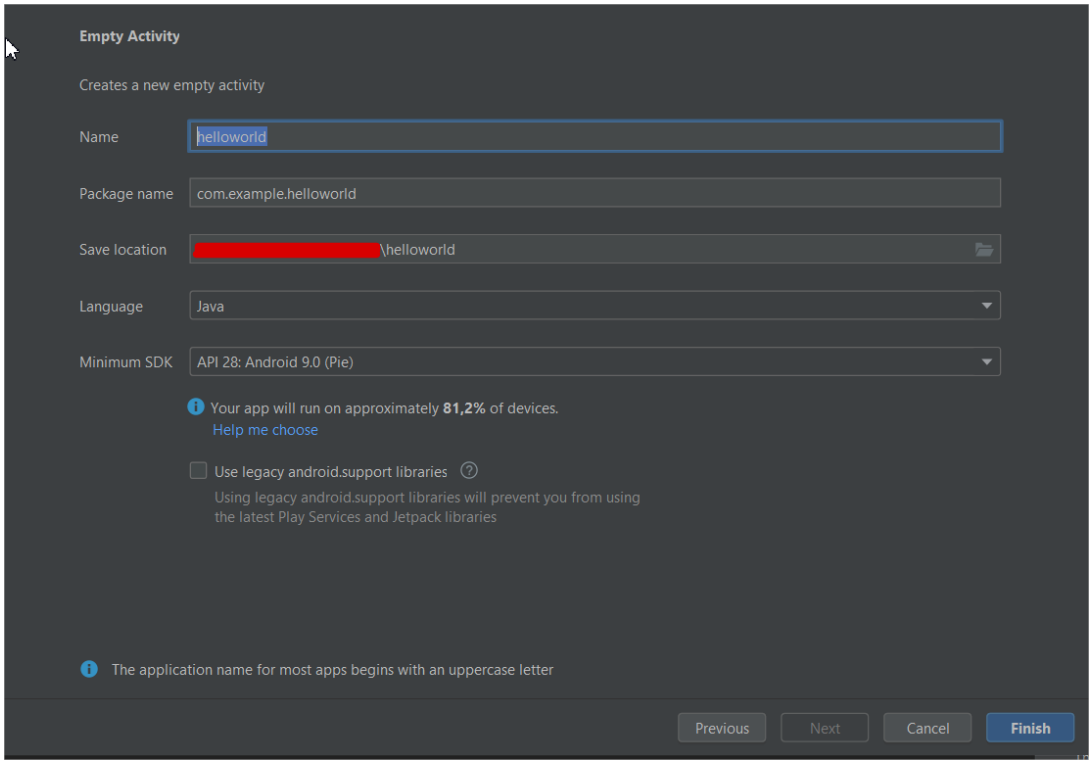
1. In Android studio go to File -> New -> New Project…



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



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


Empty Activity
Creates a new empty activity
Name
helloworld
Package name
com.example.helloworld
Save location
[redacted]\helloworld
Language
Java
Minimum SDK
API 28: Android 9.0 (Pie)
Your app will run on approximately 81,2% 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
The application name for most apps begins with an uppercase letter
Previous
Next
Cancel
Finish

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"*:

➢ 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:

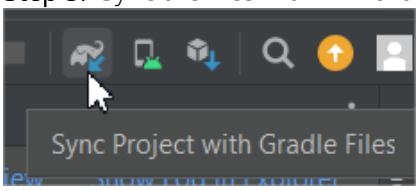
```
buildscript {
    repositories {
        google()
        mavenCentral()
        jcenter()
        maven { url "https://jitpack.io" }
        // to read credential local file
        def gradlecredentials = "credentials.gradle"
        if (project.file(gradlecredentials).exists()){
            apply 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{
                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:2.4+'
}
```

Step 3: Sync the files within Android Studio by clicking on the button :



: in case you find the error *"Build was configured to prefer settings repositories over project repositories(...)"* it means there are redundant definitions in the settings.gradle file.

Clean so that only remains

```
rootProject.name = "helloworld"
include ':app'
```

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)

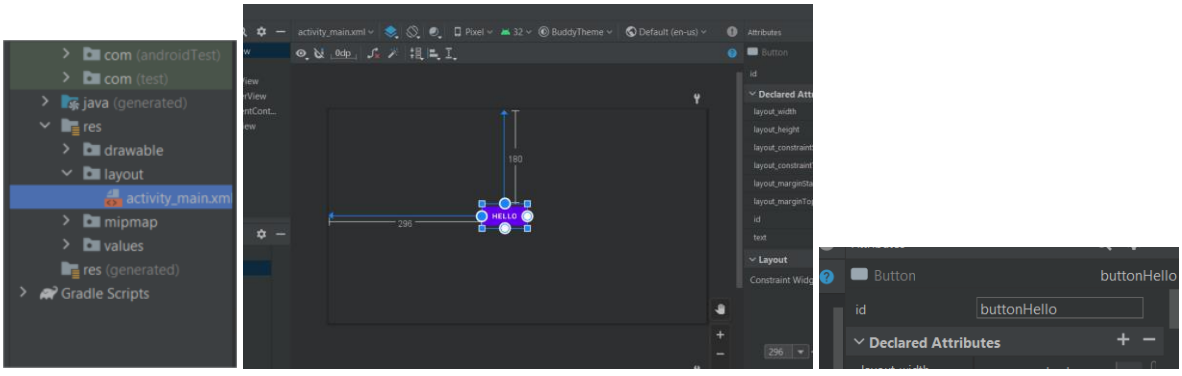
```
<uses-permission android:name="com.bfr.buddy.resource.GUI" />
```

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

Step 1 : In the xml code of your layout, add the following “android:id “ information :

```
android:id="@+id/view_face"
```

Step 2 : Insert a *Hello* button in the layout, and call it 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;
```

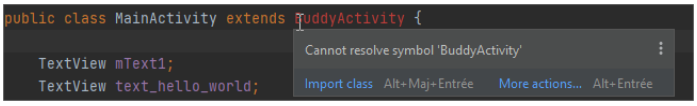
Secondly, you need to extend your MainActivity class with **BuddyActivity** (instead of the default AppCompatActivity)

```
public class MainActivity extends BuddyActivity
```

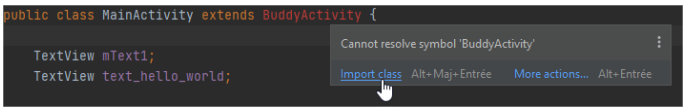
Tips:

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

Place your mouse on BuddyActivity



*Import **class***



*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)**

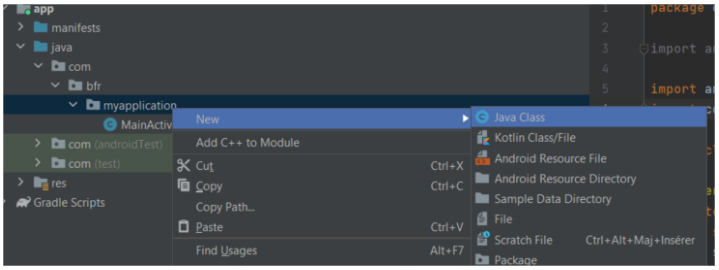
```
// in onSDKReady
BuddySDK.UI.setViewAsFace(findViewById(R.id.view_face));
```

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

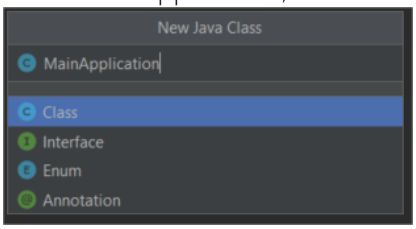
```
// set the button click listener
findViewById(R.id.buttonHello).setOnClickListener(
view -> BuddySDK.Speech.startSpeaking("Hello"));
```

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">
```

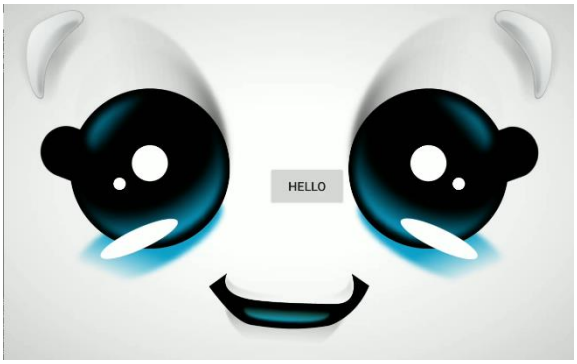
1. At the end, your manifest should look like this:

```
<?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:



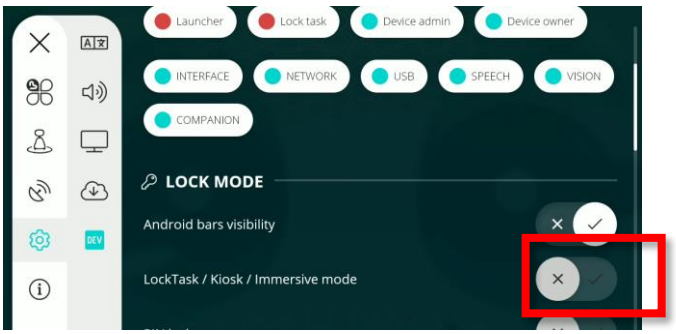
4 - IMPORTANT NOTE ON THE LOCKTASK MODE

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



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 

- When asked for a pin code, enter the 6 last digits of your robot IMEI and validate
  ➔ The dev mode should now be active

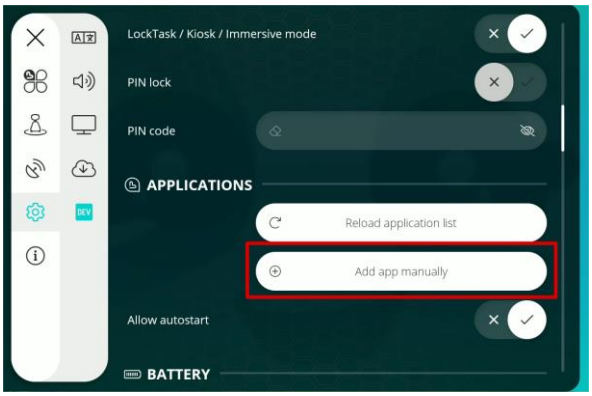
When in production, we highly recommend to enable the Locktask mode and disable the dev mode. As a consequence, your app will have to be whitelisted if you decide to Locktask your robot.

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

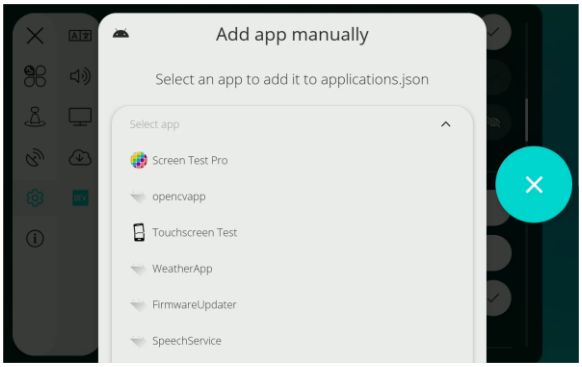
5 - OPEN AN APP FROM THE BUDDYCORE MENU

*** The lazy way (experimental, only from the dev menu)

You can add your app from the “Dev” tab in dev mode



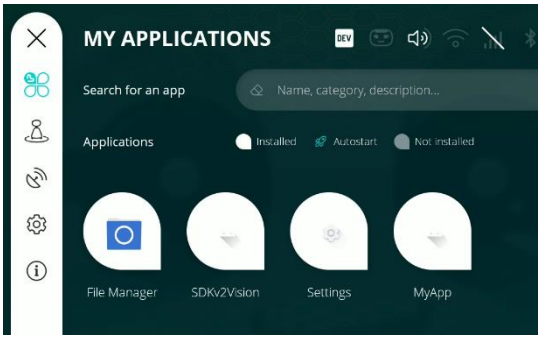
Select your app from the list



Then reboot the robot to take the changes into account.

*** The full complete way:

(The following is for BuddyOS>=1.4.x , for older version, please consult the legacy section at the end)

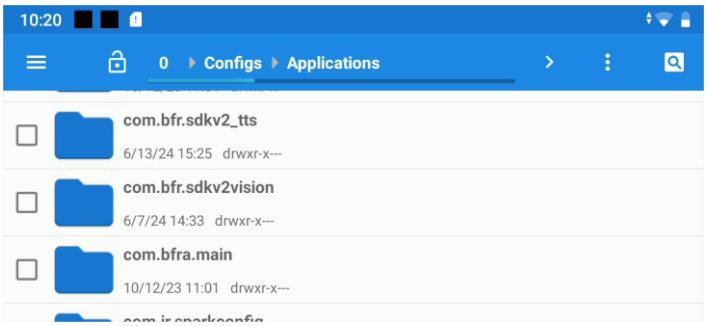


This section explains how to add your app (or any app) to the BuddyCore *My Applications* menu.
This has two main effects:

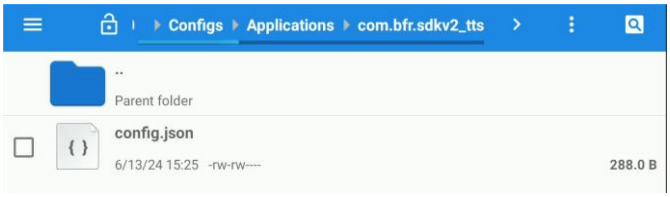
- You will be able to lauch your app from the menu (obviously)
- The app will be in the whitelist if you want to put your Buddy in Locktask
  (see previous chapter)

*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 separate folders located in /sdcard/Configs/Applications



Each folder is named after the package name of the app, and contains on config.json file



Below is the non exhaustive list of paramereters in this json file, and their defaut value:

{

"AskToQuit": true, *// with or w/o a confirmation dialog when closing the app*

"Autostart": false, *// autostart with BuddyCore*

"CloseWidgetVisibility": "ALWAYS", *// display the floating Close button from Buddy OS. It can be*

"ALWAYS": the close button will always be displayed above your app in the corner

"ON_TOUCH" : the close button will be displayed above your app in the corner, when the user touches Buddy's face, and will disappear after a few seconds

"NEVER" : the close button is always hidden

"MenuWidgetVisibility": "ALWAYS", *// display the floating Menu button from Buddy OS. It can be*

"ALWAYS": the Menu button will always be displayed above your app in the corner

"ON_TOUCH" : the Menu button will be displayed above your app in the corner, when the user touches Buddy's face, and will disappear after a few seconds

"NEVER" : the Menu button is always hidden

"Name": "My app", *// The displayed name in the Menu*

"NotificationsWidgetVisibility": "ALWAYS", *// display the floating notification button from Buddy OS. It can be*

"ALWAYS": the Menu button will always be displayed above your app in the corner

"ON_TOUCH" : the Menu button will be displayed above your app in the corner, when the user touches Buddy's face, and will disappear after a few seconds

"NEVER" : the Menu button is always hidden

"Package": "*the.name.of.the.package*",

"ShowInMenu": true *// Show or not in the menu*

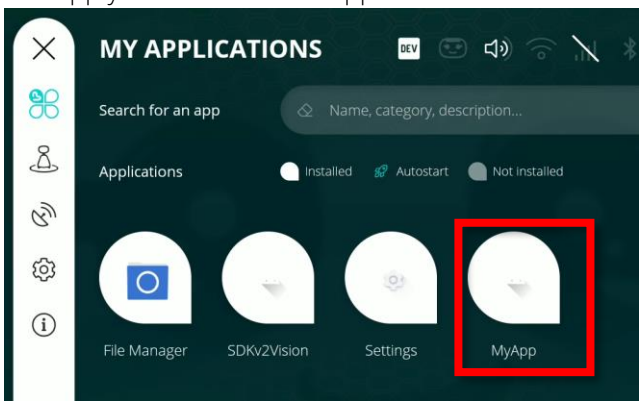
"DevOnly": false, *// The app is visible in the menu only when the dev mode is enabled*

}

In short, to whitelist your app and access it from the menu, you have to :

1) Create a folder in /sdcard/Configs/Applications named after the package name of your app
2) Create a config.json file inside containing the above parameters, with the desired value
3) reboot

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 has an autonomous mode called "Companion". This mode is also the internal engine that starts and stop your app. 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
- To be able to pause/resume your apps, you should not use the tag "android:launchmode= singleInstance" in your activity manifest

- Do not call any SDK-related function before the onSDKReady() callback has been called.
- Do not call any SDK-related function in the onPause or onStop callback of your activities, as the SDK might not be available anymore.

- 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.

With all that said, the following section explains how to integrate your app within the Companion mode of Buddy.

7 - Integrate your app with Companion

Companion is Buddy autonomous mode. Continuously, it executes various tasks (also called *missions*), following a specific trigger.

Starting your app is one of those tasks.

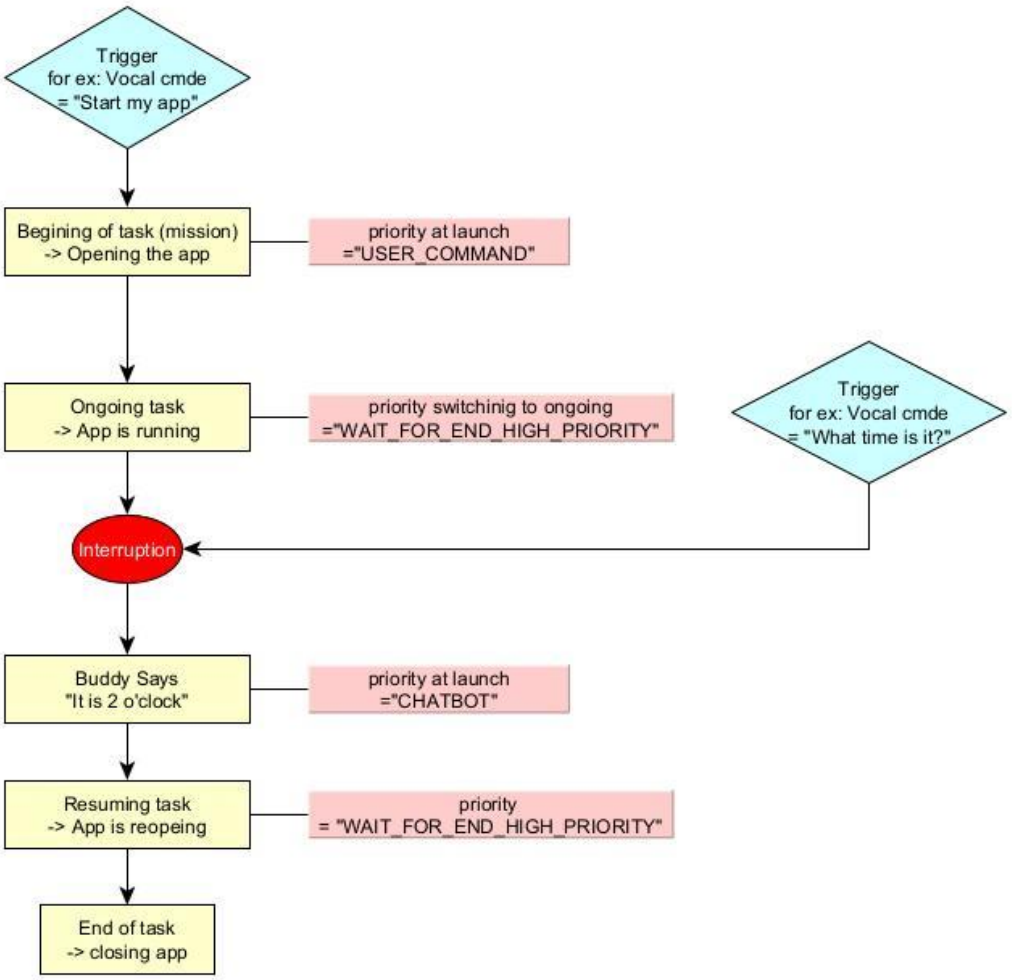
For instance, here is a non exhaustive list of the available tasks with their respective triggers:

- Touch on the head -> play an animation (BI)
- Touch on the mouth -> start listening
- Vocal command to walk around -> random stroll
- Click on an icon in the menu -> open an app
- ...

Those task are executed and kept in memory in an internal mermory stack. Hence, a task can be interrupted by another and resume depending of its assigned priority. Only a task with a higher priority can interrupt another one. The latter can be defined as resumable (stackable) or not.

Here is the possible priorities from the lowest level to the highest:

```
AUTONOMY_LOW_PRIORITY,
WAIT_FOR_END_OF_USER_CMD_LOW_PRIORITY,
AUTONOMY_HIGH_PRIORITY,
NOTIFICATION,
WAIT_FOR_END_OF_USER_CMD_HIGH_PRIORITY,
USER_COMMAND,
CHATBOT,
WAKE_UP,
EMMERGENCY
```



Sequence of execution of a task and an interruption

To fully integrate your app within Companion, you will have to specify the priorities and other parameters within a .json file called a *domain*.

The default domain json file looks like this:

```
{
 "missions": [

  {
   "name": "Open the app X",                    //Name of the task
   "speechTriggers": [                          //Vocal commands to start the task
    "Please open my application",
    "Start the app X"
],
  "priority": "USER_COMMAND"                    // priority of the task to execute it
   "waitForEndOfPriority": "WAIT_FOR_END_OF_USER_CMD_HIGH_PRIORITY",   // priority during the ex-
ecution
   "killCurrent": true,                 // wether it keeps the interrupted task in  memory
   "stackable": true,                   // wether this task can be resumed after  an interruption
    "task": "runActivity(my.package.name), // the actual task to open your app
    "startFromMenu": true           // Should be TRUE for an app
  }
 ]
}
```

The task of opening your app is defined by "task": "runActivity(my.package.name).

By default its priority at launch is “USER_COMMAND”, wich is recommended for anything requested by the user.

By default the priority during the app execution is lowered to "waitForEndOfPriority": "WAIT_FOR_END_OF_USER_CMD_HIGH_PRIORITY”. This allows the other tasks with higher priorities to be executed and pause your app. For instance, you can trigger Buddy, ask something by voice, and at the end of the vocal interaction, your app will resume.

You app can resume because because the parameter "stackable": true.

Please note that opening your app will interrupt and make Buddy forget what it is currently doing with the parameter "killCurrent": true.

All those parameters can be modified if you consider it doesn’t fit well with your app.

In short what you have to do to specify those parameters:

1) Create a .json file and name at you convenience (for instance “myapp.json”)
2) Fill in the file with the above parameters
3) Copy this file on the robot in /sdcard/Configs/Users/Default/Companion/Domain/ (create the folders if hey don’t exist)
4) Reboot the robot

A few Usecases, where you'd need to modify the default parameters

** Make the app ignore or not the strokes&pats

You might already noticed that Buddy reacts to touches on the head and body. Behind the curtain, it actually execute a task of playing an animation with priority "AUTONOMY_HIGH_PRIORITY".

As a consequence, if you want to allow those reactions while your app is running, you have to lower its ongoing priority with

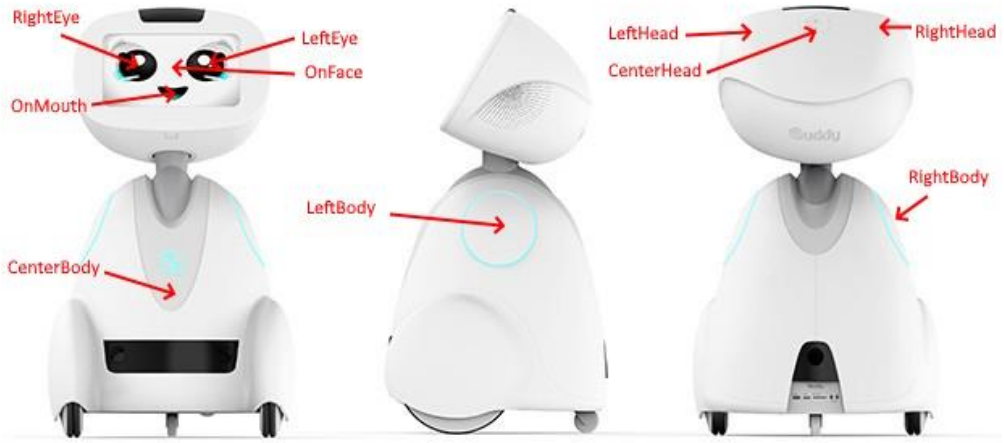
"waitForEndOfPriority": "WAIT_FOR_END_OF_USER_CMD_LOW_PRIORITY"

On the contrary, if you don't want your app to be disturbed with those touch reactions, you have to leave the ongoing priority untouched i.e.

"waitForEndOfPriority": "WAIT_FOR_END_OF_USER_CMD_HIGH_PRIORITY"

: this will bypass all the touch interactions. If you want to enable some and diasble others, you will have to pick and choose in your code with the api BuddySDK.Companion.raiseEvent("disableXXX")

Where XXX can be (self explanatory) : OnRightEye, OnLeftEye, OnFace, OnMouth, OnCenterBody, OnLeftBody, OnRightBody, OnLeftHead, OnRightHead, onCenterHead



** Specify a vocal command to start your app

You can specify any vocal command that will start your app here:

```
speechTriggers": [                    //Vocal commands to start the task
      "Please open my application",
      "Start the app X",
       " Any other custom vocal command"
],
```

 : it will only work with an internet connection, as it uses the GoogleSpeech API to transcribe the vocal command

 : remember the vocal transcription doesn't work with exotic app names

 : Please insure that there are not any conflict with existing vocal command.

** Make the app ignore The "OK Buddy" Vocal Trigger

To do so, you will have to raise the ongoing priority to at least

"waitForEndOfPriority": "WAKE_UP"

On the contrary, if you want your app to be paused by the vocal trigger, you have to leave the ongoing priority untouched i.e.

"waitForEndOfPriority": "WAIT_FOR_END_OF_USER_CMD_HIGH_PRIORITY"

 : for BuddyOS >=1.5, any app that would access directly the microphone (a recording app, a Zoom-like video-chat app,...) must have a "waitForEndOfPriority": "WAKE_UP", to by pass the trigger and access the audio stream.

** Specify an activity to start

You can apss an initla activity as an argument to the runActivity task:

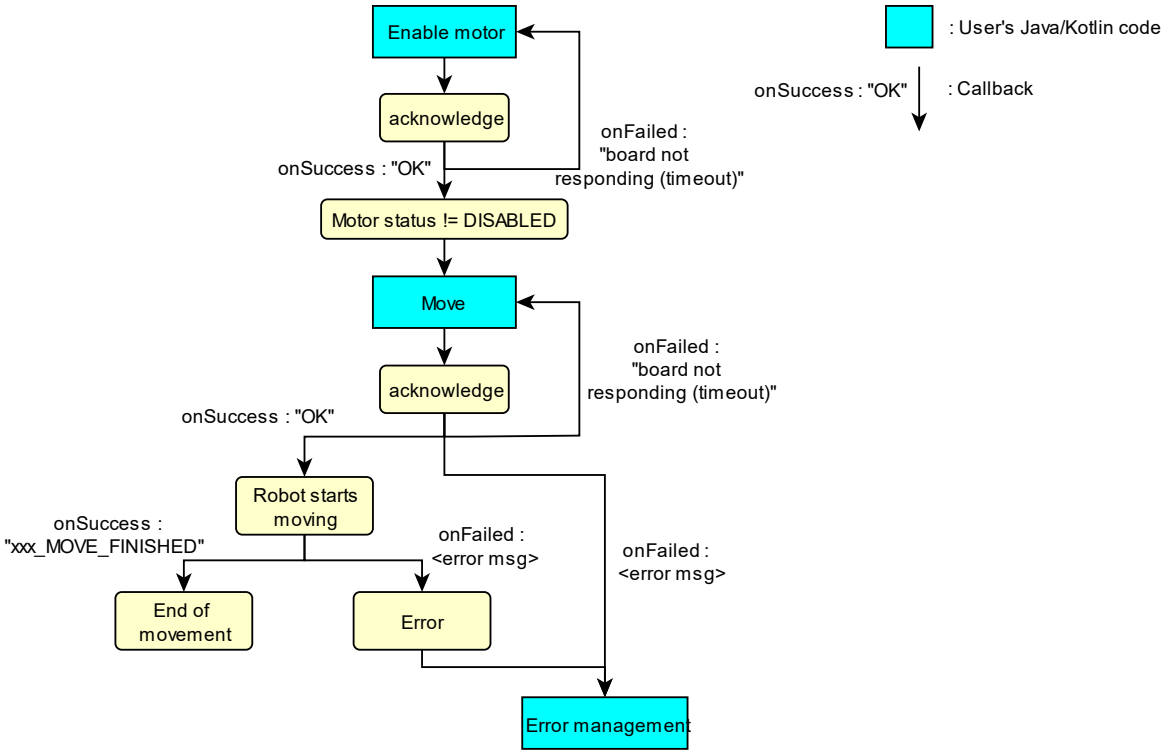
```
"task": "runActivity(com.name.of.package, com.name.of.package.initial.activity)
```

8- SDK API DOCUMENTATION

1. ACTUATORS

Sending a command to the actuators usually returns an acknowledge signal via a callback.

So here is the typical workflow, when using Buddy's motors:



1) Enable the motor : just after calling the respective function (for instance *EnableWheels()*) you should receive the "OK" acknowledgement. If not, retry.
2) When the motor is enabled, you should check that its status is not "DISABLED" anymore
3) Move : Call the respective function (for instance *BuddySayYes()*)
4) Just after calling the function, you should receive an onSuccess:"OK" acknowledgement. It means the instruction is taken into account and the robot will start moving.
   If you receive the specific onFailed : "board not responding (timeout)" ack signal, it means the system did not succeed to take into account your instruction, and you might want to try again.
   In the case of another error, you will have to manage it in your code.
5) As soon as the instruction is processed by the system, the robot starts moving.
6) At the end of the movement, you should receive an onSuccess:"xxx_MOVE_FINISHED" ack.signal
7) In case of error, you will receive an onFailed+error message signal , and you will have to manage the error in your code.

NB: Sending two consecutive commands for the same movement will cancel the first one.

- 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
  - "Canceled when it is interrupted by another command
  - "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
  - "Canceled when it is interrupted by another command
  - "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*, *BuddyAcceleration*, *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
- **BuddyAcceleration [optional]: can be :**
  - BuddyAccelerations.LOW (0.3 m/s²)
  - BuddyAccelerations.NORMAL (0.5 m/s²) BY DEFAULT
  - BuddyAccelerations.HIGH (1 m/s²)
  - A positive value in m/s² (0 = no limitation)
- **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

***** The following is available with BuddyOS >=v1.4

*void USB.*moveBuddy(*LinearSpeed, AngularSpeed, Distance, Buddy Acceleration, RspCallback*)

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

Params:

- LinearSpeed (float): linear speed you want Buddy to move - in m/s (-0.56 m/s - 0.56 m/s)
- AngularSpeed (float) [optional]: angular speed you want Buddy to move - in deg/s (-180 °/s - 180 °/s)
- Distance[optional] (float): give the distance to reach in meter
  If absent, Buddy will move indefinitely at the given speed
- BuddyAcceleration [optional]: can be :
  - BuddyAccelerations.LOW (0.3 m/s²)
  - BuddyAccelerations.NORMAL (0.5 m/s²) BY DEFAULT
  - BuddyAccelerations.HIGH (1 m/s²)
    A positive value in m/s² (0 = no limitation)
- RspCallback (IUsbCommadRsp): return
  - "OK" when launch
  - "WHEEL_MOVE_FINISHED" when the move is finished
  - "NOK" when fail

void rotateNoPrecision(Speed, Angle, MaxAcceleration, TaskCallback)

Purpose: Rotate and stop without deceleration

Params:

- Speed: speed you want Buddy to rotate in °/s (should be > 0)
- Angle: angle you want Buddy to rotate in ° (- : counterclockwise, + clockwise)
- MaxAcceleration: maxAcceleration - maximum authorised acceleration in m/s² - (should be >=0 - 0 = no limitations)
- TaskCallback: callback to the task created by this function

void goStraightNoPrecision(Speed, Distance, MaxAcceleration, TaskCallback)

Purpose: Rotate and stop without deceleration

Params:

- Speed: speed you want Buddy to move in °/s (should be > 0)
- Distance: distance you want Buddy to travel in m (+ : forward, - backward)
- MaxAcceleration: maxAcceleration - maximum authorised acceleration in m/s² - (should be >=0 - 0 = no limitations)
- TaskCallback: callback to the task created by this function

void setBuddySpeed(LinearSpeed, AngularSpeed, MaxAcceleration, TaskCallback)

Purpose: Make Buddy move at a given linear and angular speed.

linear and angular speeds are saturated in firmware such that wheels can't turn too fast, but ratio between linear and angular speed is ensured

Params:

- linearSpeed: linear speed you want Buddy to move - in m/s (-0.56 m/s - 0.56 m/s)
- angularSpeed: angular speed you want Buddy to move - in rad/s (-1.4 rad/s - 1.4 rad/s)
- MaxAcceleration: maxAcceleration - maximum authorised acceleration in m/s² - (should be >=0 - 0 = no limitations)
- TaskCallback: callback to the task created by this function

- 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

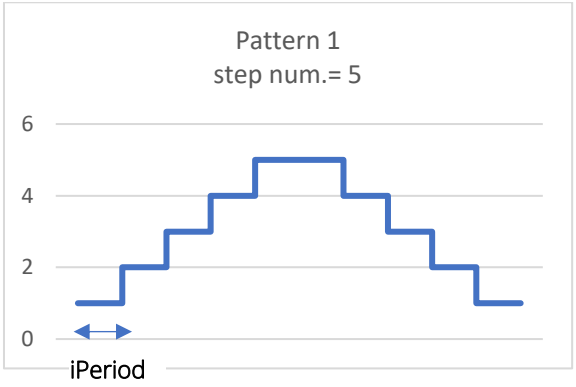
void USB.updateAllLedWithPattern(iColor, iPattern, iPeriod, iStep, iRspCallback)
Purpose: Switch all Leds with a pattern with the color we choose.

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

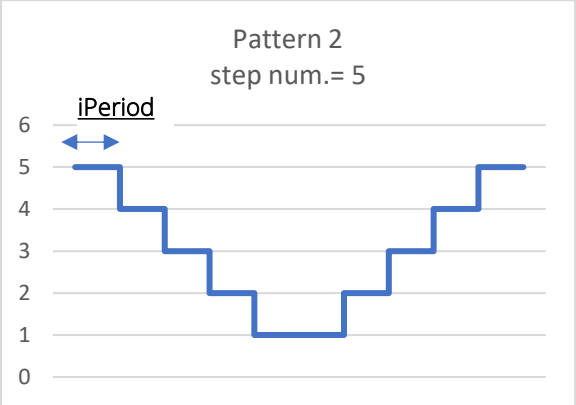
Pattern 1:

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



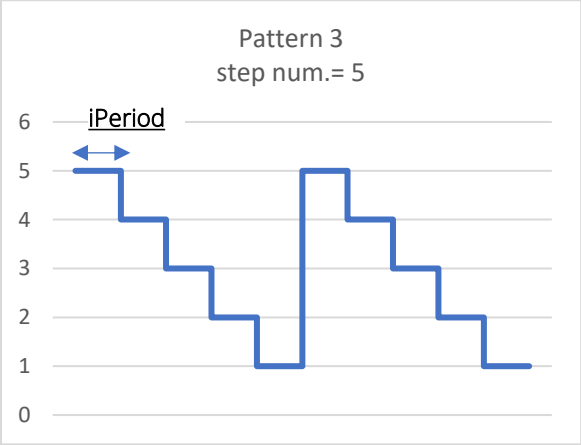
Pattern 2 :

Same as before but beginning with the LED ON.



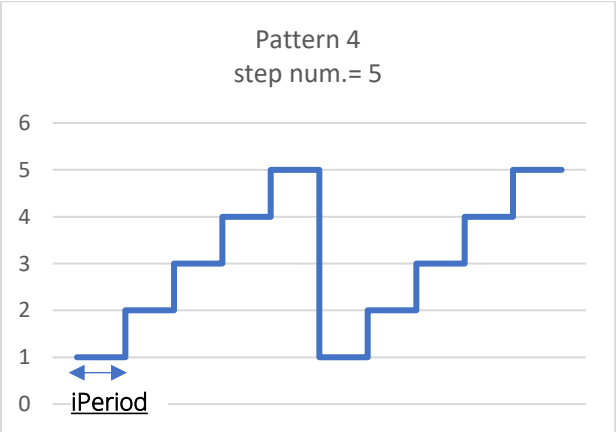
Pattern 3 :

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**

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:

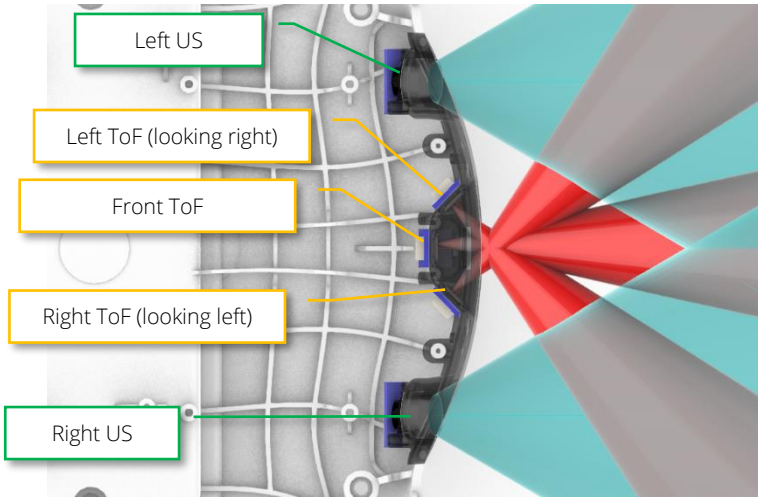
```
BuddySDK.Sensors.BodyTouchSensors().Torso().isTouched();
BuddySDK.Sensors.BodyTouchSensors().LeftShoulder().isTouched();
BuddySDK.Sensors.BodyTouchSensors().RightShoulder().isTouched();
```

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

*boo*lean Sensors.Battery().isCharging()

Purpose: Check if the battery is in charge

Return:

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

 The actuator and sensor infos are automatically updated every ~100ms. In addition, you can access all the above informations manually via a callback. Some additional infos are available that way.

```
// callback to get infos from Sensor/Motor board every 100ms
IUsbAidlCbListener usbCbk = new IUsbAidlCbListener.Stub() {
  @Override
  public void ReceiveMotorMotionData(MotorMotionData motorMotionData) throws RemoteException {
    motorMotionData.leftWheelCurrent; // electrical current in mA consumed by the motor, useful to detect a peak of effort
    motorMotionData.leftWheelMode; // motor mode (enable, disable, ...)
    motorMotionData.leftWheelSpeed; // current speed of the motor
    motorMotionData.last10LeftWheelSteps; //stores the last 10 steps made by the motor during the last 100ms
    // hence, each step represent around 10ms. Sum to know the wheel displacement. A whole wheel rotation = 840 steps

    // same thing for the right wheel
  }


  @Override
  public void ReceiveMotorHeadData(MotorHeadData motorHeadData) throws RemoteException {
    motorHeadData.noCurrent; // current in mA consumed by the motor, usefull to detect a peak of effort
    motorHeadData.noMode; //motor mode (enable, disable, ...)
    motorHeadData.noPosition; // position of the head

    //same thing for the Yes motor
  }

  @Override
  public void ReceiveHeadSensorData(HeadSensorData headSensorData) throws RemoteException {
    // head sensors
  }

  @Override
  public void ReceiveBodySensorData(BodySensorData bodySensorData) throws RemoteException {
    // body sensors
  }

  @Override
  public void ReceivedVocalData(VocalData vocalData) throws RemoteException {
    vocalData.AmbientSoundLevel; // sound level in dB
    vocalData.SoundSourceLocalisation; // direction of the sound in °
    vocalData.TriggerScore; //score of the recognized "OKBuddy"
  }
};
```

Then you will have to suscribe to the callback when the SDK is ready:

```
// suscribe to Sensor/Motor boards infos
//should be updated every ~100ms
BuddySDK.USB.registerCb(usbCbk);
```

Unsuscribe when exiting:

```
// when exit, unsuscribe
BuddySDK.USB.unRegisterCb(usbCbk);
```

2. 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)

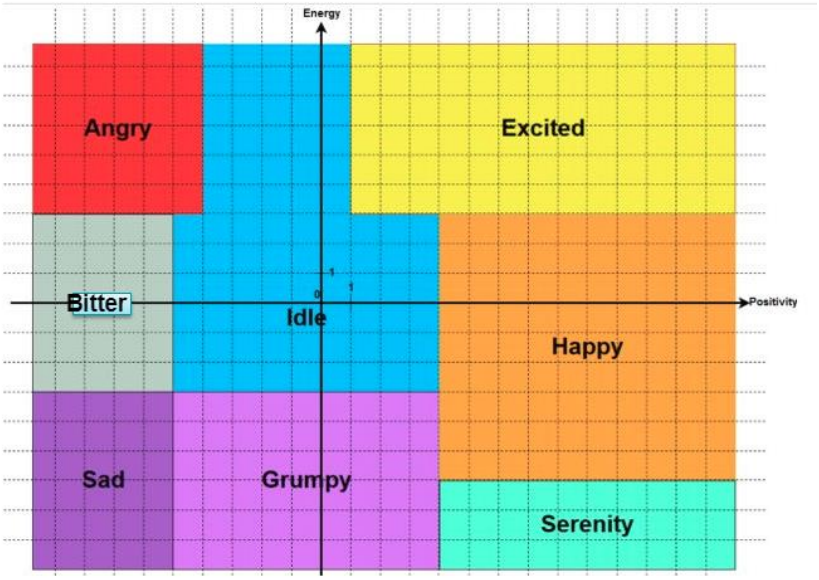
Purpose: Play different gimmicks

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***)

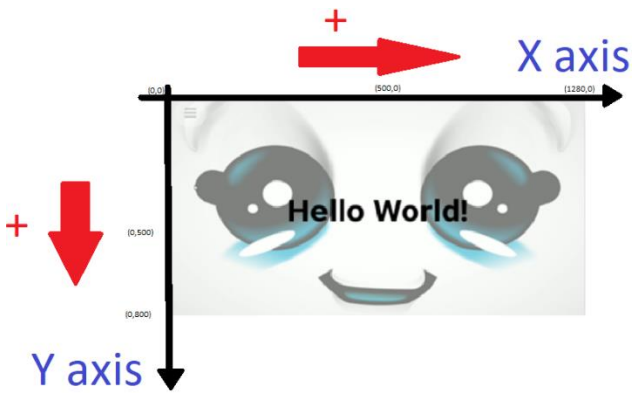
Purpose: Play face animation relative to the current expression
Params:

- **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. Value between [0;1300]
- **iY (float):** Vertical axis. Value between [0;900]
- **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)

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

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

void removeFaceTouchListener(IUIFaceTouchCallback iCallback)

- **Purpose** : 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

3. 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 american female voice
- For BuddyOS >=1.4
- **"amir" :** Arab male voice
- **"yasmin" :** Arab female voice
- **"lena" :** German female voice
- **"max" :** German male voice
- **"alice" :** British female voice
- **"mark" :** English american male voice
- **"lola" :** Spanish female voice
- **"manuel" :** Spanish male voice
- **"elisa" :** Italian female voice
- **"show" :** Japanese male voice
- **"alex" :** Dutch male voice
- **"guus" :** Dutch 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 the 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):**
  - 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

4. 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

*** For BuddyOS > =1.4, please use the following instead

ArucoMarker[] getListOfArucos(ArucoDictionary)

Returns :

An array of ArucoMarker containing:

- The id of the marker
- The four corners coordinates X,Y (floats[4][2]) in % of the width and height - the first corner is on the the top-left, then the others follow in the clockwise direction The list of score (= confidence) of the detections

Params :

- [Optional] dictionary : the dictionary of marker to detect

  cf.https://docs.opencv.org/4.x/de/d67/group__objdetect__aruco.html#ga4e13135a118f497c6172311d601ce00d

  By default= DICT_APRILTAG_36h11 =20, for the Buddy playing cards.

Pose estimateArucoPose(ArucoMarker, arucoSize)

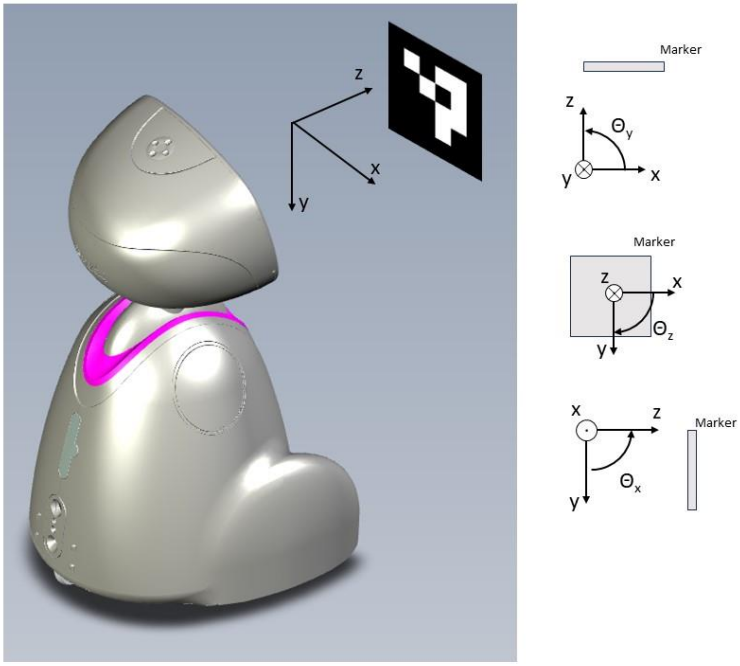
Purpose: Give the Pose(translation/rotation) of a marker from the camera center.

Returns :

a Pose containing the X, Y, Z distance, and ThetaX,ThetaY,ThetaZ angles

Params :

- ArucoMarker : a marker previously detected with getListofAruco
- arucoSize (double) : the size of the marker in reality. The unit (cm, mm, m,...) defines the unit of the output



QRCode [] getListOfQRCodes(QRDetectionMethod)

Purpose: Detect and decode the QRcodes in front of the camera

Returns :

An array of QRCodes containing:

- The content of the QRCode
- The four corners coordinates X,Y (floats[4][2]) in % of the width and height - the first corner is on the the top-left, then the others follow in the clockwise direction The list of score (= confidence) of the detections

Params :

- [Optional] QRDetectionMethod: should be:
  - QRDetectionMethod.FAST: fast detection
  - QRDetectionMethod.NORMAL: average speed but more accurate (detection at 1-2m) ,

- QRDetectionMethod.HIGH_PRECISION : slow but can detect a QRCode at ~3-4m, however, it cannot read the QRcode content and doesn't give accurate corners coordinates. Thus it cannot be used for pose estimation

Pose estimateQRCodePose(QRCode, qrSize)

Purpose: Give the Pose(translation/rotation) of a marker from the camera center.

Returns :

a Pose containing the X, Y, Z distance, and ThetaX,ThetaY,ThetaZ angles

Params :

- QRCode: a QRCode previously detected with getListofQRCodes
- qrSize (double) : the size of the QRcode in reality. The unit (cm, mm, m,...) defines the unit of the output

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

void Vision. stopMotionDetection ()

Purpose: stop the motion detection process

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

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

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

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.

❖ Face recognition (SDK>2.3)

This algorithm allows to recognize a person from its face. Basically, it is done in two steps:

1. Enrolment: taking a reference picture of a face and saving a label (typically the name or a nickname) that is linked to the person's face
2. Recognition : being able to give the corresponding label whenever a similar face is seen again

Here are a few guidelines regarding this algorithm:

- The face must not touch the borders of the image (we have to be sure the entire face is visible for an optimal recognition). Actually it must be far from 3% of the border of the image
- The face should be big enough during the enrolment for an optimal recognition. We recommend the size face to be at least 13% of the whole image (face width > 13% image width).
- You should always insure that the person is static and the image is stable, to avoid noise, motion blur, flickering,…
- The more reference pictures you take, the better the recognition will be; you might want to take pictures of various configuration (facing, profile, looking up, from up-close, from afar,…)
- However, the number of reference picture should be roughly the same between individuals
- Just after saving the reference picture of the face, the recognition score should be >0.8. So a good rule of thumb would be : make recognition just after, if the score is not high enough, save it again.
- The actual reference picture is not saved during the enrollement. If for some reason you want to access it in the future, you will have to save it yourself using the getCVResultFrame() method

void Vision. saveFace(Detections faces, int idx, String faceName, String fileName, IVisionRsp visionCbk)

Purpose: Save a face(identity) for further recognition

Params :

- faces (Detection) : array of face detections, typically obtained from the detectFace() method
- idx (int) : index of the face you want to save
- facename (String) : a 'name' or identity to label the face with
- [optional] filename (String) : the complete path of the file where to save the face and corresponding name/lable on the device. Should be a *.bin file
  - By default: /storage/emulated/0/identities/identities.bin
- visionCbk (iVisionRsp) : IVision.Stub() that returns:
  - onSuccess : when the face has been successfully saved
  - onError: otherwise

 : This method goes hand in hand with a face detection. So typically the workflow would be:

3. Make a face detection
4. From all the found faces, select the one you want to save, with the criteria of your choice (ex: take the face the most on the left side, using getLeftPos())
5. Save that face

 : By default the faces/labels are saved in a binary file in : /storage/emulated/0/identities/identities.bin . It means, by default, it will be shared with any other app using the SDK. If you don't want that, be sure to give a specific filename+path as an argument. That way your app will be using its own a set of faces to recognize.

void Vision. saveFace(float left, float right, float top, float bottom, String faceName, String fileName, IVisionRsp visionCbk)

Purpose: Save a face(identity) for further recognition. It takes the coords of a bounding box containing the face. In case of multiple faces present in the bounding box, only the first one is taken into account.

Compared to saveFace(Detections faces, int idx, String faceName, String fileName, IVisionRsp visionCbk), this function is useful if you want to use your own face detection algorithm or if you already know the location of the face you want to save.

Params :

- left (float) : coordinate of the left border of the bbox in % of the image width
- rightt (float) : coordinate of the right border of the bbox in % of the image width
- top (float) : coordinate of the top border of the bbox in % of the image height
- bottom (float) : coordinate of the bottom border of the bbox in % of the image height
- facename (String) : a 'name' or identity to label the face with
- [optional] filename (String) : the complete path of the file where to save the face and corresponding name/lable on the device. Should be a *.bin file
  - By default: /storage/emulated/0/identities/identities.bin
- visionCbk (iVisionRsp) : IVision.Stub() that returns:
  - onSuccess : when the face has been successfully saved
  - onError: otherwise

void Vision. loadFaces(String fileName, IVisionRsp visionCbk)

Purpose: Load list of saved faces from a previous call of saveFace().

Params :

- [optional] filename (String) : the complete path of the file where the faces/labels have been saved on the device. Should be a *.bin file
  - By default: /storage/emulated/0/identities/identities.bin
- visionCbk (iVisionRsp) : IVision.Stub() that returns:
  - onSuccess : when the faces list of faces has been successfully loaded
  - onError: otherwise

 : By default the faces/labels are saved in a binary file in : /storage/emulated/0/identities/identities.bin . It means, by default, it will be shared with any other app using the SDK. If you don't want that, be sure to give a specific filename+path as an argument. That way your app will be using its own a set of faces to recognize.

void Vision. getSavedNames ()

Purpose: Get the list of saved faces (corresponding labels) after a call of loadFaces().

Returns : a list of String containing all the saved faces/associated labels

void Vision. removeFace(int idx, String fileName, IVisionRsp visionCbk)

Purpose: Remove a saved face from the list of saved faces

Params :

- idx (int) : the index of the face to remove
- [optional] filename (String) : the complete path of the file storing the faces. Should be a *.bin file
  - By default: /storage/emulated/0/identities/identities.bin
- visionCbk (iVisionRsp) : IVision.Stub() that returns:
  - onSuccess : when the faces list of faces has been successfully deleted
  - onError: otherwise

 : WARNING, this action is irreversible .Although a backup file is automatically created in the default folder, you might want to add an extra safety and create your own backup first.

FaceRecognition Vision. recognizeFace (Detections faces, int idx, String faceName)

Purpose: Recognize a face in the image

Params :

- faces (Detection) : array of face detections, typically obtained from the detectFace() method
- idx (int) : index of the face you want to save

Returns : a FaceRecognition object containing the name and the score of recognition (between [0;1])

: This method goes hand in hand with a face detection. So typically the workflow would be:

1. Make a face detection
2. From all the found faces, select the one you want to save, with the criteria of your choice (ex: take the face the most on the left side, using getLeftPos())
3. Recognize that face

FaceRecognition Vision. recognizeFace (float left, float right, float top, float bottom)

Purpose: Recognize a face in the image. It takes the coordinates of a bounding box containing the face. Compared to recognizeFace (Detections faces, int idx,), this function is useful if you want to use your own face detection algorithm or if you already know the location of the face you want to recognize.

Params :

- left (float) : coordinate of the left border of the bbox in % of the image width
- rightt (float) : coordinate of the right border of the bbox in % of the image width
- top (float) : coordinate of the top border of the bbox in % of the image height
- bottom (float) : coordinate of the bottom border of the bbox in % of the image height

Returns : a FaceRecognition object containing the name and the score of recognition (between [0;1])

FaceRecognition[] Vision. getTopKResults (int k)

Purpose: After calling the face recognition method, get the top k-results, starting from the most recognized face.

Params :

- k (int) : the number of candidates

Returns : a list of FaceRecognition objects containing the names and the scores of recognition (between [0;1])

❖ Get the frame from the camera

Bitmap Vision. getGrandAngleFrame()

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

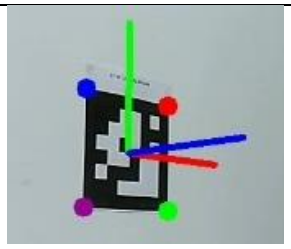
Returns : the camera frame as a Bitmap of 1024x768 pixels

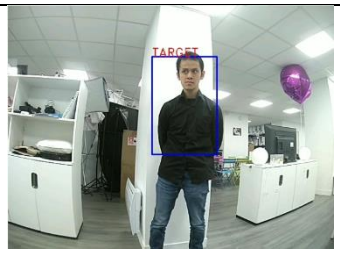
Bitmap Vision. getCVResultFrame()

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

Returns : a Bitmap of 1024x768 pixels

Examples of image returned by each CV algorithm:

Aruco Marker detection	
Aruco marker pose estimation	
QRCode detection	
QRCode Pose estimation	
Face Detection	

Facial recognition	
Human detection	
Motion Detection	
Visual Tracking	

5. Behaviour Instructions (BI)

The following is for the SDK >=v2.3 . For older versions, please see the deprecated documentation in Annex.

To read a BI, which contains sequential or parallel instructions to play robot behaviours, you must first create a task like this :

```
Task biTask = BuddySDK.Companion.createBITask(source.getAbsolutePath(), exoView, imgView);
```

The prototype of the function is this :

Task createBITask(String iPath, final PlayerView playerView, final ImageView imageView, boolean addDebugLogAsIntermediateResult=false, Array<String> permissions, Map<String, String> biParameters)

- iPath : path to the BI. If only the name is provided the interpreter will search the file in the folder BI/Custom then in BI/Behaviour.
- playerView : the PlayerView of the application (used to play videos). Optional
- imageView : the imageView of the app (used to show pictures). Optional
- AddDebugLogAsIntermediateResult: Boolean to indicate if we should ping debug information in the intermediate result callback of the task. Optional
- Permissions: List of permissions accorded for this task (ex: "com.bfr.buddy.resource.SPEECH", ...). Put null if you want to forward the permissions of your activity to the task. Optional
- biParameters: [OPTIONAL] Mapping of parameters <String, String> where the first string is the name of the parameter, and the second is the parameter value in string format

To run the BI you can use the method start from the Task object you got earlier. It needs a TaskCallback as argument.

```
biTask.start(new TaskCallback() {
    @Override
    public void onStarted() {
    }
    @Override
    public void onSuccess(@NonNull String s) {
    }
    @Override
    public void onCancel() {
    }
    @Override
    public void onError(@NonNull String s) {
    }
    @Override
    public void onIntermediateResult(@NonNull String s) {
    }
}) ;
```

TaskCallback is an interface that defines several functions to implement that are called at different time during the execution of the BI:

- OnStarted: called when the BI starts running
- onIntermediateResult: If the variable AddDebugLogAsIntermediateResult has been set to true when creating the task, it will be called : when the instruction will be executed just after the delay, and when the instruction has finished the execution. The string that is given in parameter is Start/Stop instruction: <name of instruction>_<id number of instruction>
- onSuccess : called when the BI has been fully executed
- onCancel : called when the BI has been cancelled
- onError : called when there is an error during the execution. The string in parameter contains the error message

To stop the BI just call the stop function like this :

```
biTask.stop();
```

To run a BI by category you need to use this method:

Task createBITaskCategory(String iCategory, final PlayerView playerView, final ImageView imageView, boolean addDebugLogAsIntermediateResult=false, Array<String> permissions, Map<String, String> biParameters)

- iCategory: category of the bi to play
- playerView : the PlayerView of the application (used to play videos). Optional
- imageView : the imageView of the app (used to show pictures). Optional
- AddDebugLogAsIntermediateResult: Boolean to indicate if we should ping debug information in the intermediate result callback of the task. Optional
- Permissions: List of permissions accorded for this task (ex: "com.bfr.buddy.resource.SPEECH", ...). Put null if you want to forward the permissions of your activity to the task. Optional
- biParameters: [OPTIONAL] Mapping of parameters <String, String> where the first string is the name of the parameter, and the second is the parameter value in string format

The interpreter will search a BI at random in the custom and the standard folder with the category provided.

It returns a Task that is used the same way as the other method.

The categories usable are the following:

Angry, Awake, BadAnswer, BlinkDouble, BlinkLeft, BlinkRight, CenterHead, CenterHeart, CliffBack, CliffFront, CliffLift, Congratulations, Dance, Defeat, Demo, DemoShort, DetectSound, DoctorCall, 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

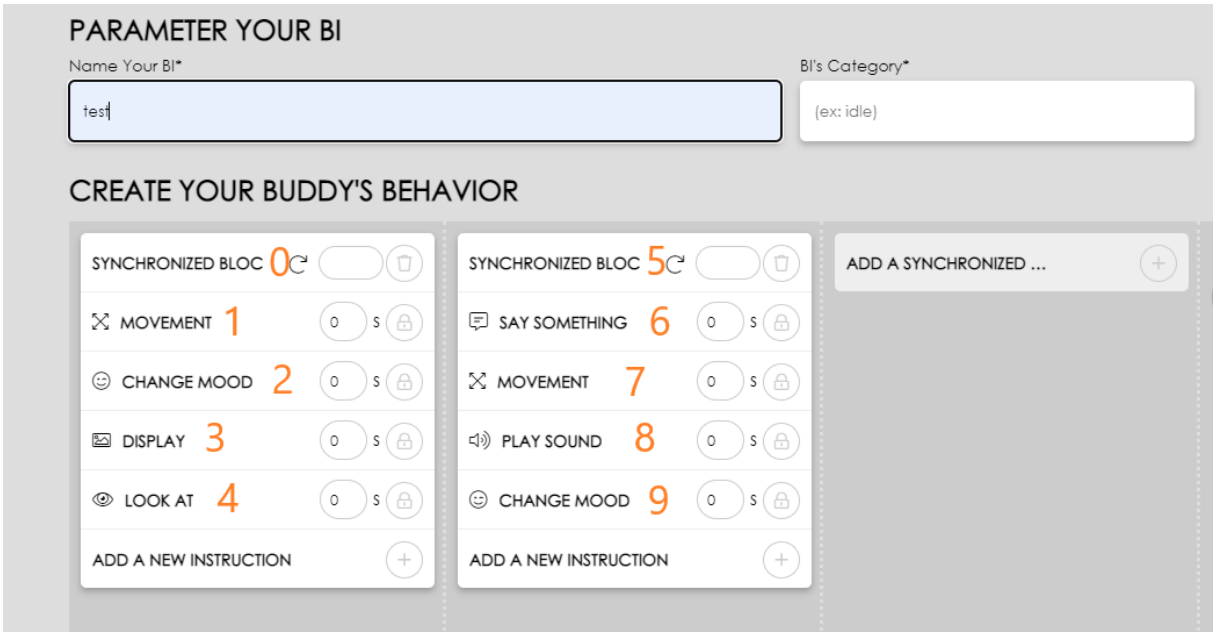
Some instructions will search the resources in specific folder:

The RunScriptInstructionBi will search the BI first in the BI/Custom folder then in BI/Behaviour.

The DisplayImageInstruction will search pictures in the folder Pictures and videos in the folder Videos.

The PlaySoundBehaviourInstruction will search the sound file in the Music folder.

When a BI is executed each instruction get an id. It begins by 0 and is incremented for each instruction following. This picture presents an example of id assignation :



If we launch the precedent BI without error and without cancelling it the onIntermediateResult callback will be called twice for each instruction. If we log the string in parameter we can get the following:

Start instruction: SynchronizedBehaviourInstruction_0

Start instruction: MovementBehaviourInstruction_1

Stop instruction: MovementBehaviourInstruction_1

Start instruction: SetMoodBehaviourInstruction_2

Stop instruction: SetMoodBehaviourInstruction_2

...

Start instruction: SetMoodBehaviourInstruction_9

Stop instruction: SetMoodBehaviourInstruction_9

Stop instruction: SynchronizedBehaviourInstruction_5

Example to read a BI:

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(Environment.DIRECTORY_DOWNLOA
DS).toString();

    String fileName = docPath + "/" + biName;

    File source = new File(fileName);



    try {

    Task biTask = BuddySDK.Companion.createBITask(source.getAbsolutePath(),
exoView, imgView, true);
biTask.start(new TaskCallback() {

    @Override
    public void onStarted() {
        Log.d(TAG, "bi started");
    }

    @Override
    public void onSuccess(@NonNull String s) {
        Log.d(TAG, "bi success "+s);
    }

    @Override
    public void onCancel() {
        Log.d(TAG, "bi cancelled");
    }

    @Override
    public void onError(@NonNull String s) {
        Log.e(TAG, "bi error "+s);
    }

    @Override
    public void onIntermediateResult(@NonNull String s) {
        Log.e(TAG, "intermediate result "+s);
    }
});

    } catch (Exception e) {

         e.printStackTrace();

    }

}
```

In this example the BI xml must be in the download folder in Android to work and you must have a videoview and imageView in your layout application. If you put your BI in BI/Custom you can put only the name of the Bi in the createBITask function.

 : all the sounds played from the BI are located in the folder:

/storage/emulated/0/Music/BFR

Feel free to use it in your app!

6. User interface (UI)

void *UI*.setCloseWidgetVisibility(FloatingWidgetVisibility iVisibility);

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

void *UI*.setNotificationsWidgetVisibility(FloatingWidgetVisibility iVisibility)

Purpose: set the visibility of the Notifications Button in your app. This button is managed by BuddyCore and is always hidden by default if application is running, but visible in Companion mode. 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[4]

[2] The activity you touch must have called setViewFace

[3] The activity you touch must have called setViewFace

[4] The activity you touch must have called setViewFace

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?"

void UI.showNotification(StringMultilang iTitleSecondary, StringMultilang iText, int iAutoHideTime, boolean iEphemeral, byte[] iImage, NotificationButtonGroup iButtons, NotificationCompanionTask iSwipeAction);

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

Params:

- lTitleSecondary (StringMultilang): A second title on top of a notification. The second title is always a name of an application.
- lText (StringMultilang): A text of a notification.
- lAutoHideTime (int): Time in seconds after which notification will be automatically hidden. 5 seconds by default.
- lEphemeral (boolean): Ephemere notifications are not added into “not seen" list.
- iImage (byte[] or int): Byte array or resource ID for an image to show on the right side of a notification. Image on the left side is always an icon of an application.

- IButtons (NotificationButtonGroup): Group of up to three buttons for the bottom of a notificaiton. Each button has a predefined color, predefined or custom text and a NotificationCompanionTask which will be executed when button is clicked.
- ISwipeAction (NotificationCompanionTask): Task to be executed when a notification is swiped out before being hidden automatically.
- StringMultilang: a text object which holds translations. Can be formed from <Locale, String> pairs or resource ID. If there is no localised string, you can also create this object using only a simple string.
- NotificationButtonGroup: Group of buttons for notificaiton (up to three buttons). Also has convenience subclasses NotificationButtonGroupOK (has one OK button) and NotificationButtonGroupYesNo (provides two predefined buttons)

void UI.hideNotification();

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

 : 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.

7. Companion (SDK>=v2.3)

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.

Params:

- 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.iCallback (Consumer<Boolean>): True if code correct, false if code incorrect or closed.

Task Companion.createTask(String nameOfTheTaskToCreate, TaskParameters taskParameters, String[] permissions, Func onFinished)

Purpose: Create a task object that represent the task you want to launch. You can start and stop the task object.

Params:

- nameOfTheTaskToCreate (String): name of the task you can launch
- taskParameters (TaskParameters): parameters needed by some tasks to run, use an empty TaskParameters if the task doesn't need parameters, else create map <String, String> and put this map in the TaskParameters()
- permissions (String[]): permissions needed by the task to run, for example the task random stroll needs to move the Head, wheels, vision, sensors so you will need to add those permissions in a String[].
  (cf this section for the details on the permissions)
- onFinished (void Func): function called when the task is finished

• Task Companion.createFollowMeTask(FollowMeMode followmeMode, Long interval, String[] permissions)

Purpose: Create a FollowMe task object that represent the task FollowMe you want to launch. You can start and stop the task object. If the FollowMeMode is WATCH_ME the robot will follow the user with the eyes, the head and the body but will stay at the same place, If the FollowMeMode is FOLLOW_ME the robot will do the same and additionally the robot will move to follow you wherever you go.

Params:

- followmeMode (enum): You can use FollowMeMode.WATCH_ME or FollowMeMode.FOLLOW_ME to define which of the two tasks you want to use.

- interval (long): interval of time between each eye blinking, in milliseconds
- permissions (String[]): permissions needed by the task to run, for example the task random stroll needs to move the Head, wheels, vision, sensors so you will need to add those permissions in a String[].
  (cf this section for the details on the permissions)

- Task Companion.createGivenDayTask(String[] permissions)

Purpose: Create a GivenDay task object that represent the task GivenDay you want to launch. You can start and stop the task object.

Params:

- permissions (String[]): permissions needed by the task to run, for example the task random stroll needs to move the Head, wheels, vision, sensors so you will need to add those permissions in a String[].
  (cf this section for the details on the permissions)

- Task Companion.createGivenHourTask(String[] permissions)

Purpose: Create a GivenHour task object that represent the task GivenHour you want to launch. You can start and stop the task object.

Params:

- permissions (String[]): permissions needed by the task to run, for example the task random stroll needs to move the Head, wheels, vision, sensors so you will need to add those permissions in a String[].
  (cf this section for the details on the permissions)

- Task Companion.createRandomStrollTask(String[] permissions)

Purpose: Create a randomStroll task object that represent the task RandomStroll you want to launch. You can start and stop the task object. The robot will do a random stroll.

Params:

- permissions (String[]): permissions needed by the task to run, for example the task random stroll needs to move the Head, wheels, vision, sensors so you will need to add those permissions in a String[].
  (cf this section for the details on the permissions)

Once the task created is started:

- Task.start(ITaskCallback) : starts the defined Companion Task
  - With ITaskCallback :
    - OnSuccess: is called when the task is finished successfully. The string value is the the value returned by the task and the effect is the effect applied to the world state.
    - OnStarted: called when the task start

- onCancel: is called when the task is finished because it was cancelled.
- The onError is called when the task is finished because of an error. The string tells you error.
- onIntermediateResult : function called when the task has an intermediate result

➢ ITask.stop() : stops the defined Companion Task

Tasks available in companion launched by createTask

.

- ## Task Realign

Companion.createTask("realign", taskParameters, permission)

"realign" is the name of the companion's task you want to launch from your application. The task will move the head to the angle 0 for the NO and your specified angle for the angle YES, then the robot will do a rotation to search for an apriltag in front of him and then stop the task or make a rotation of an offset angle.
We chose a speed of 30F for the rotation to be sure to have the less motion blur possible and be able to detect the April tag without problem. The April tag should be placed at 4 meters at max. if the distance is superior to 4 meters Buddy will have issues to detect the April tag. The April tag must be from the dictionary 4x4 .
taskParameters is a TaskParameters which takes a Map<String, String> in his constructor. You can start by creating a Map<String, String> params = new HashMap<>() and then TaskParameters taskParameters = new TaskParameters(params). The map params is here for to add specific parameters to the task realign.

Params:

- aprilTagID (String): the ID of the aprilTag you want to search around buddy. If you don't put this parameter, by default Buddy wll search for an April tag with the ID 1.
- numberRotation (String): the number of rotations you want to do with Buddy. By default, the number of rotations is 1.
- angleOffset (String): this the angle that Buddy will do after the detection of the April tag. By default, the offset angle is 0°.
- yesPosition (String): this is the angle on the motor YES that buddy will do before starting the rotation. By default, the angle is 10°.

permissions (String[]): permissions needed by the task to run, for example the task Realign needs to move the Head, wheels, vision so you will need to add those permissions in a String[].
(cf this section for the details on the permissions)

For example, if you want to search for the April tag 10, to place the head of Buddy at 20 in Yes position, to do 5 rotations and to do a 180° angle when you found the April tag you can do :

params.put(TaskSignature.Realign.aprilTagID, "10");

params.put(TaskSignature.Realign.numberRotation, “5”);

params.put(TaskSignature.Realign.wheelsOffset, “180”);

params.put(TaskSignature.Realign.yesPosition, “-10”);

Then you can use Companion.createTask(“realign”, taskParameters, permission) (cf this section for the details on how to use the createTask on your application).

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.

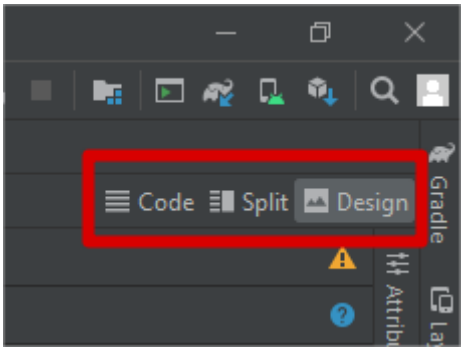
```
<Button
    android:id="@+id/button_advance"
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:text="Advance"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.209"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

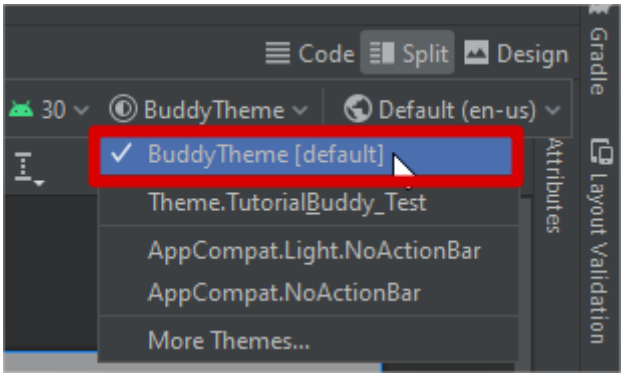
The "**Enable wheels**" button is used to, once it's clicked on, enable the wheels motor.

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!

```
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

```
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 )
```

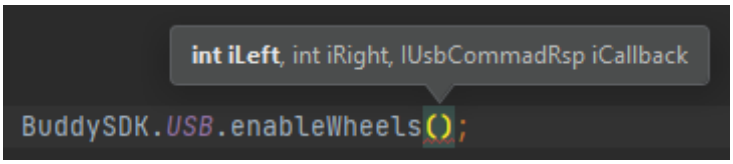
In the onCreate location, link the two buttons with the two created in the layout file.

```
//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.



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.

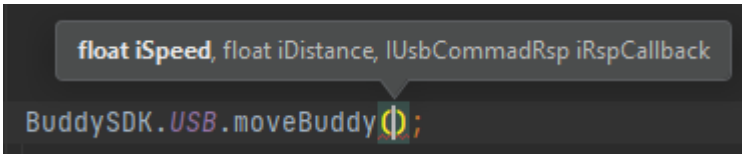
```
        BuddySDK.USB.enableWheels(turnOnLeftWeel, turnOnRightWeel, new IUsbCommadRsp.Stub() {

            @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()***.



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

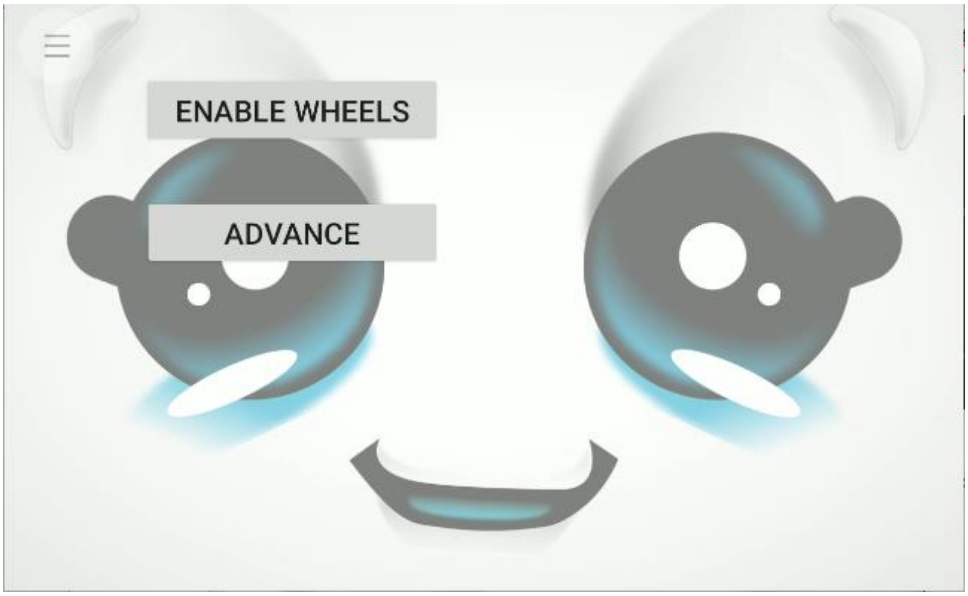
```
private void AdvanceFunct() {
    //Here we decide to go forward of 0.5meters with a 0.5m/s speed
    float speed = 0.5F;//definition of the speed (>0 to go forward , <0 to go backward)
    float distance = 0.5F;//definition of the distance to pracour(ALWAYS>0)

    //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.

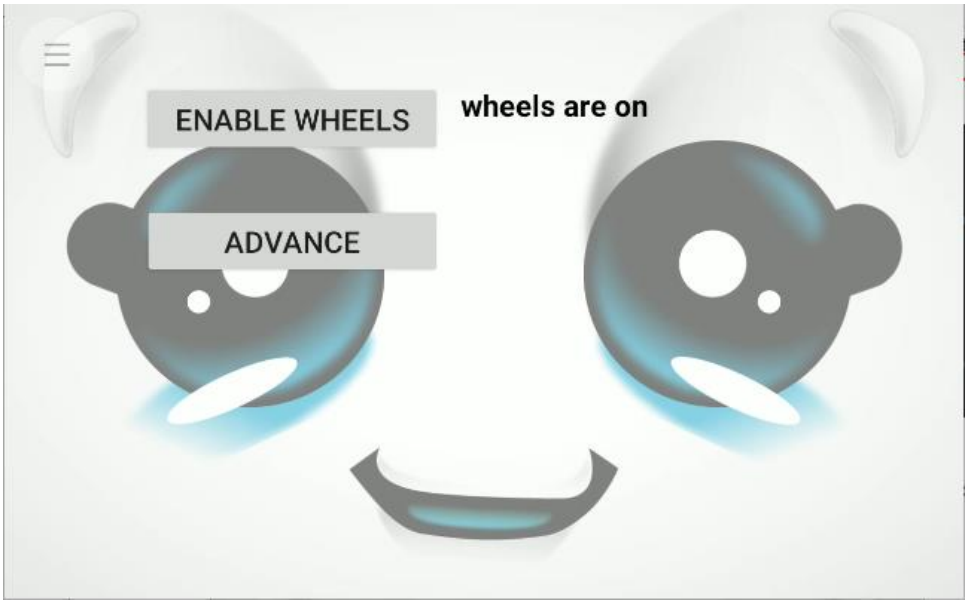
```
//call of the function to make buddy go forward or backwars
    BuddySDK.USB.moveBuddy(speed, distance, new IUsbCommadRsp.Stub() {
        @Override
        public void onSuccess(String s) throws RemoteException {
            Log.i(TAG,  msg: "AdvanceFunct: sucess");//in case of success show in the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

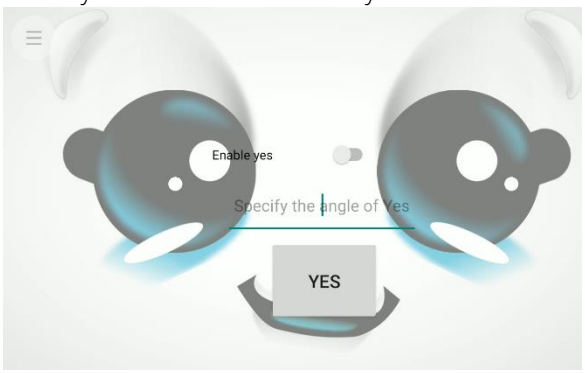
```
    }
}
```

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

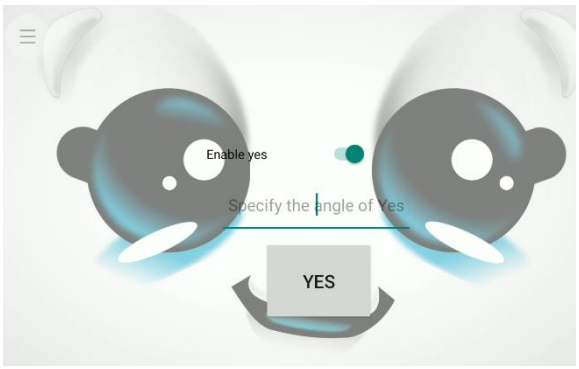
```
    //if the function did not succeed,nothing is happening
    public void onFailed(String error) {
    }
});
```

2) Running the App

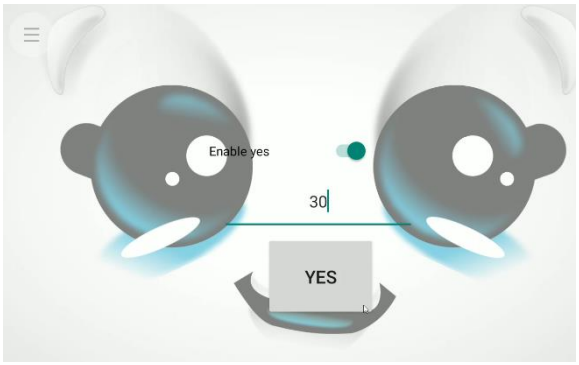
When you launch the method you can can observe a window with the face of Buddy.



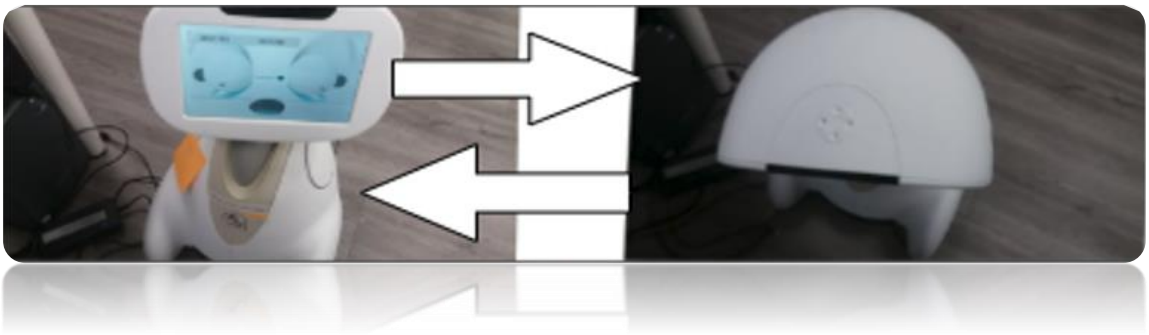
Check the switch to do a “yes move”



Enter the value of angle you want



This move is executed for BuddySayYes



3) Launch mission (Task) from Companion

In the following part, we present an example of an application which can launch a task from companion

Layout XML file :

Step 1*: Create new button to launch a task.*

Here is the button for Buddy to launch the task.

```
<Button
  android:id="@+id/btn_launch_task"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginStart="284dp"
  android:layout_marginTop="180dp"
  android:text="Launch"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent" />
```

Step 2*: Create new button to stop the task.*

Here is the button for Buddy to stop the task

```
<Button
  android:id="@+id/btn_stop_task"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginStart="284dp"
  android:layout_marginTop="28dp"
  android:text="stop"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toBottomOf="@+id/btn_launch_task" />
```

The two steps only create the UI of the two buttons, now we will link the UI with the code in MainActivity.java.

- MainActivity file :

Step1: Go to **MainActivity** and initialize the two buttons to do the start task and the stop task and link them to the two buttons created in your .xml :

```
//Create two buttons, one to start the task and on to stop the task
Button buttonStartTask = findViewById(R.id.btn_launch_task);
Button buttonStopTask = findViewById(R.id.btn_stop_task);
```

Step 2: Now you can create a Task object

```
//Create the Task
private Task task = null;
```

and initialize this object in the first button buttonStartTask with

```
task = BuddySDK.Companion.createRandomStrollTask();
```

Step 3: The object Task is now created, we can start using it.

Create a TaskCallback first that will be used to start the task.

```
//Create the task callback you'll need to put in the start
private final TaskCallback taskCallback = new TaskCallback() {

  @Override
  public void onStarted() {
    Log.i(TAG, "Task is started");
  }

  @Override
  public void onSuccess(@NonNull String value) {
    Log.i(TAG, "Task finished on success with message: " + value);
  }

  @Override
  public void onCancel() {
    Log.i(TAG, "Task finished on cancellation");
  }

  @Override
  public void onError(@NonNull String message) {
    Log.e(TAG, "Task finished on error with message: " + message);
  }

  @Override
  public void onIntermediateResult(@NonNull String s) {}
};
```

The OnStarted is called when the task starts.

The OnSuccess is called when the task is finished successfully. The string value is the value returned by the task.

The onCancel is called when the task is finished because it was cancelled.

The onError is called when the task is finished because of an error. The string tells you the error.

The onIntermediateResult is called when the task has an intermediate result. The string is the intermediate result.

In the button launch we can add the start.

```
buttonStartTask.setOnClickListener(view -> {
   //Avoid starting multiple time the task if you click many time on the button
   if (task != null)
      task.stop();
   //init the task
   task = BuddySDK.Companion.createRandomStrollTask();
   //start the task randomStrolleRandom
   task.start(taskCallback);
});
```

The part with task.stop() if the task != null needs to be used to avoid multiple instance of the task. Without this part, you would multiple instances launched of the same task and you would have problem with the face if the task use FaceEvent for example.

Finally after the creation of the task you just need to start it with task.start(taskCallback).

And add the stop in the button stop:

```
buttonStopTask.setOnClickListener (view -> {
   //stop the task randomStrolleRandom
   task.stop();
});
```

We implemented a function createTask that you can use to launch missions from SDK, but for basic mission like follow me or randomStroll we already did custom functions to facilitate your development. For now, we have createFollowMeTask, createRandomStrollTask, createGivenDayTask, createGiveHourTask and some others which will be available after. Here we chose to give you the example with the function createRandomStrollTask.

Warning : Don't forget to put the permissions needed by your application in the manifest like this :

```
<uses-permission android:name="com.bfr.buddy.resource.SPEECH" />
<uses-permission android:name="com.bfr.buddy.resource.LISTEN" />
<uses-permission android:name="com.bfr.buddy.resource.HEAD" />
<uses-permission android:name="com.bfr.buddy.resource.WHEELS" />
<uses-permission android:name="com.bfr.buddy.resource.LEDS" />
<uses-permission android:name="com.bfr.buddy.resource.SENSOR_MODULE" />
```

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

before the <application></application>. If you don't put the permissions your application will do nothing.

/*****You also manage if there is an error with the try/catch. For example if you want to create a task that need a specific permission and you forgot to add this permission in your application, you will see an error like this :

```
E  FATAL EXCEPTION: main
   Process: com.example.testcreatetask, PID: 22490
   java.lang.RuntimeException: Cannot create task followMeAndBlinkEyes because the permission com.bfr.buddy.resource.VISION is missing
```

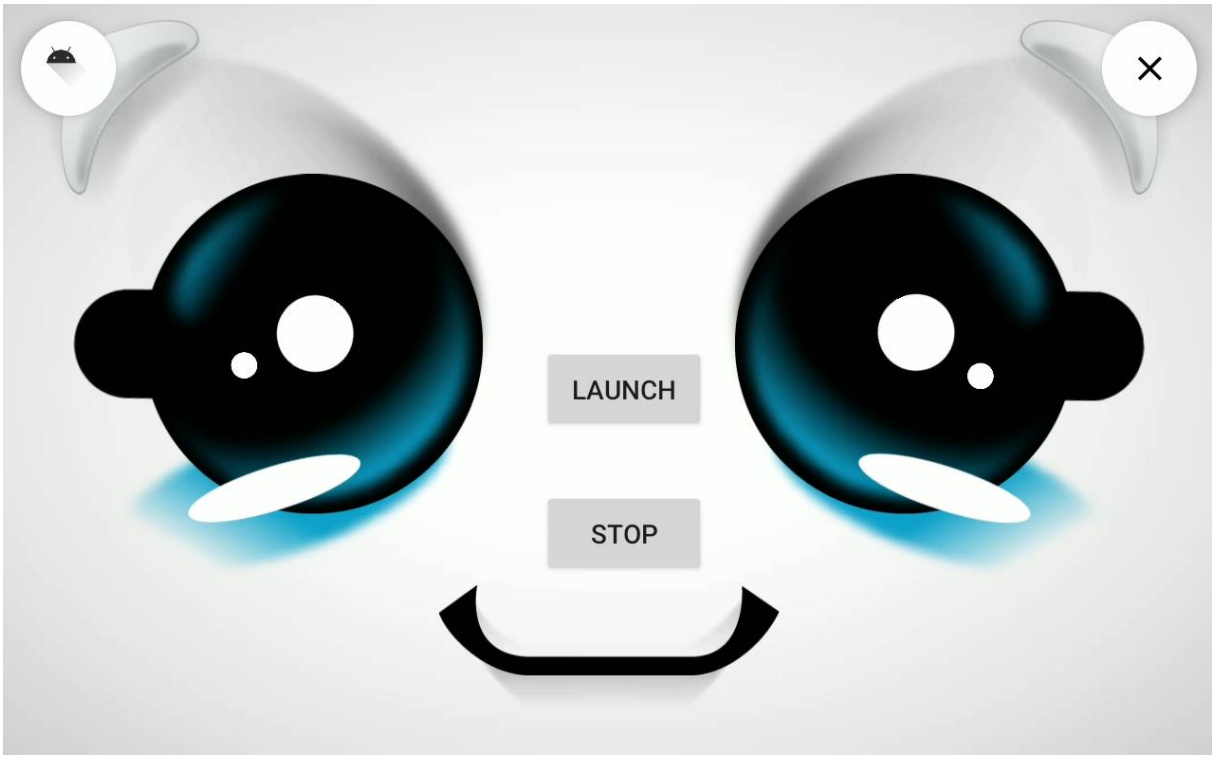
And you can see that we wanted to create a task and the permission Vision needed for the task is missing. ****/

The file should look like this at the end:

```
package com.example.testcreatetask;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import android.os.Bundle;
import android.util.Log;
import android.widget.Button;
import com.bfr.buddy.utils.events.EventItem;
import com.bfr.buddysdk.BuddyActivity;
import com.bfr.buddysdk.BuddySDK;
import com.bfr.buddysdk.services.companion.Task;
import com.bfr.buddysdk.services.companion.TaskCallback;

public class MainActivity extends BuddyActivity {

   private final String TAG = "MainActivity";

   //Create the Task
   private Task task = null;

   //Create the task callback you'll need to put in the start
   private final TaskCallback taskCallback = new TaskCallback() {

      @Override
      public void onStarted() {
         Log.i(TAG, "Task is started");
      }

      @Override
      public void onSuccess(@NonNull String value) {
         Log.i(TAG, "Task finished on success with message: " + value);
      }

      @Override
      public void onCancel() {
         Log.i(TAG, "Task finished on cancellation");
      }
```

```
    @Override
    public void onError(@NonNull String message) {
        Log.e(TAG, "Task finished on error with message: " + message);
    }

    @Override
    public void onIntermediateResult(@NonNull String s) {}
  };


  @Override
  public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Create two buttons, one to start the task and on to stop the task
    Button buttonStartTask = findViewById(R.id.btn_launch_task);
    Button buttonStopTask = findViewById(R.id.btn_stop_task);

    buttonStartTask.setOnClickListener(view -> {
      //Avoid starting multiple time the task if you click many time on the button
      if (task != null)
        task.stop();
      //init the task
      task = BuddySDK.Companion.createRandomStrollTask();
      //if you want to create a follow me task that does the watch me with 7000 ms between blink eyes
and with permission
      //from your androidManifest.xml
      //task = BuddySDK.Companion.createFollowMeTask(FollowMeMode.WATCH_ME, 7000L, null);
      //start the task randomStrolleRandom
      task.start(taskCallback);
    });

    buttonStopTask.setOnClickListener (view -> {
      //stop the task randomStrolleRandom
      task.stop();
    });
  }

  @Override
  public void onSDKReady() {
    super.onSDKReady();
  }

  @Override
  public void onEvent(EventItem iEvent) {
    super.onEvent(iEvent);
  }

}
```

And the UI of the app on the robot should look like this:


LAUNCH
STOP

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

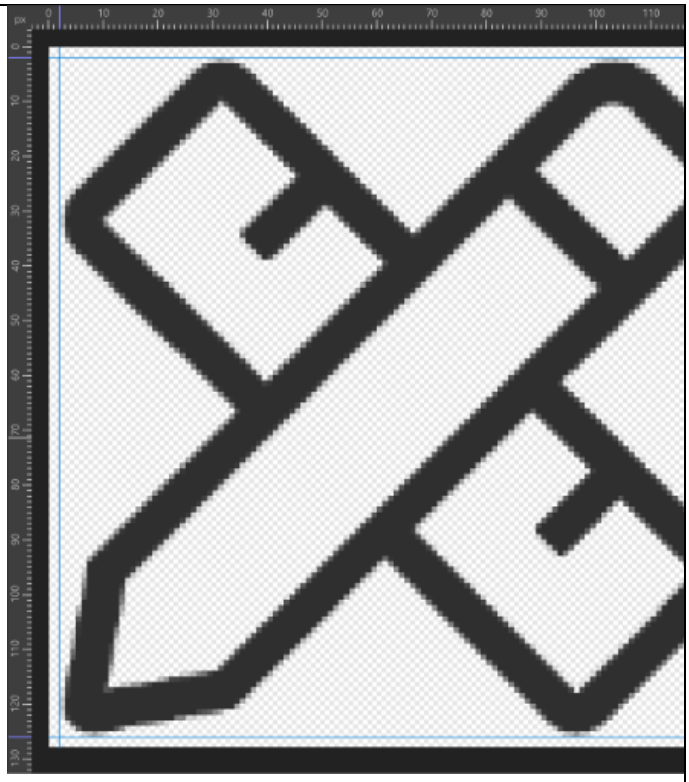
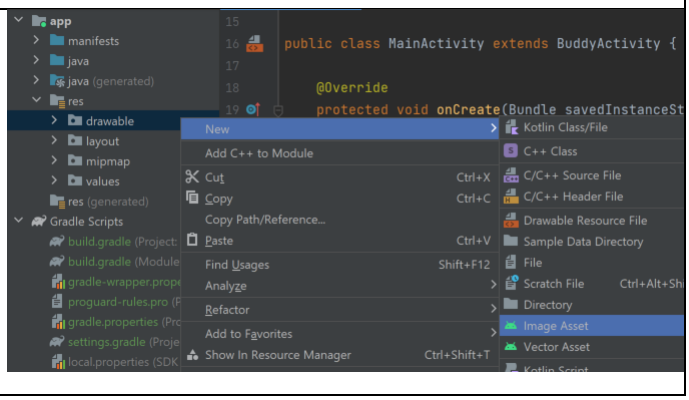
For French folder:

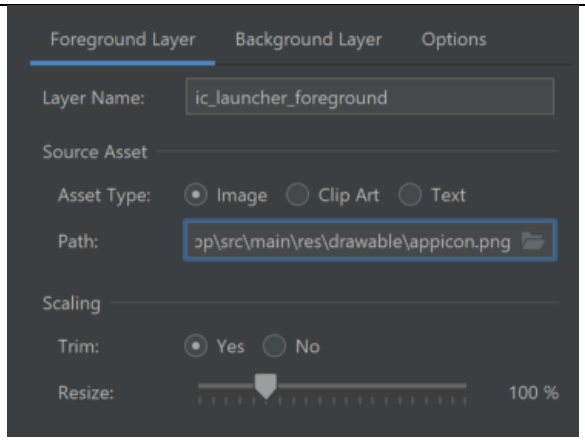
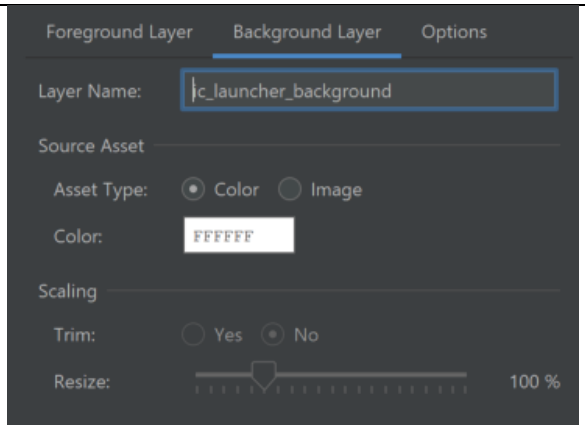
[redacted]

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**	

Set Foreground layer settings : 1. Select your image under Source Asset Path 2. Trim to Yes, Resize 100%	
Set white color as backgound	
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	

ANNEX

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.

Behaviour Instructions (BI) DEPRECATED (SDK<v2.3)

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 iAlgorithmEndListener, 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, CliffBack, CliffFront, CliffLift, Congratulations, Dance, Defeat, Demo, DemoShort, DetectSound, DoctorCall, 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 iListener)

Purpose: register to a class that implement OnRunInstructionListener

Param:

- iListener : class that implement OnRunInstructionListener

```
public void
registerOnRunInstructionListener(OnRunInstructionListener
iListener)
```

- 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
OnRunInstructionListener, 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(Environment.DIRE
CTORY_DOWNLOADS).toString();

    String fileName = docPath + "/" + biName;

    File source = new File(fileName);



    try {

        //BehaviourAlgorithmStorage storage =
serializer.read(BehaviourAlgorithmStorage.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.getAlgorithm(), this, this, imageView, videoView);

            biPlaying=true;

        }
```

```
    } catch (Exception e) {

        e.printStackTrace();

    }

}
```

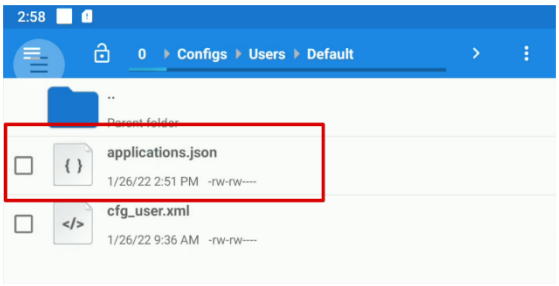
The xml must be in the download folder in Android

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

OPEN AN APP FROM THE BUDDYCORE MENU (DEPRECATED, FOR BUDDYOS <1.4.X)

Each app has its own folder.

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:

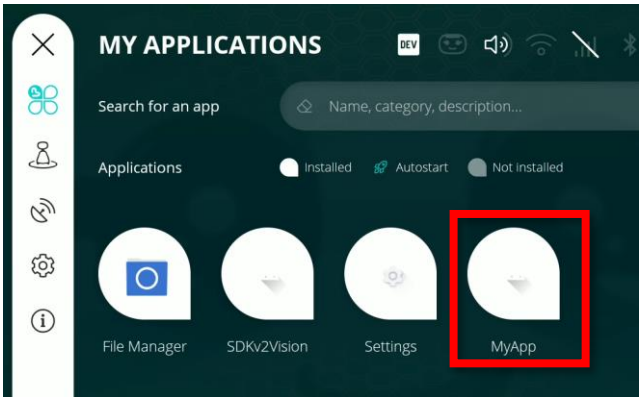
```
  {
   "Autostart": false,
   "Package": "com.android.myapplication",
   "ShowInMenu": true
 }
```

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.


MY APPLICATIONS
DEV
Search for an app
Name, category, description...
Applications
Installed
Autostart
Not installed
File Manager
SDKv2Vision
Settings
MyApp