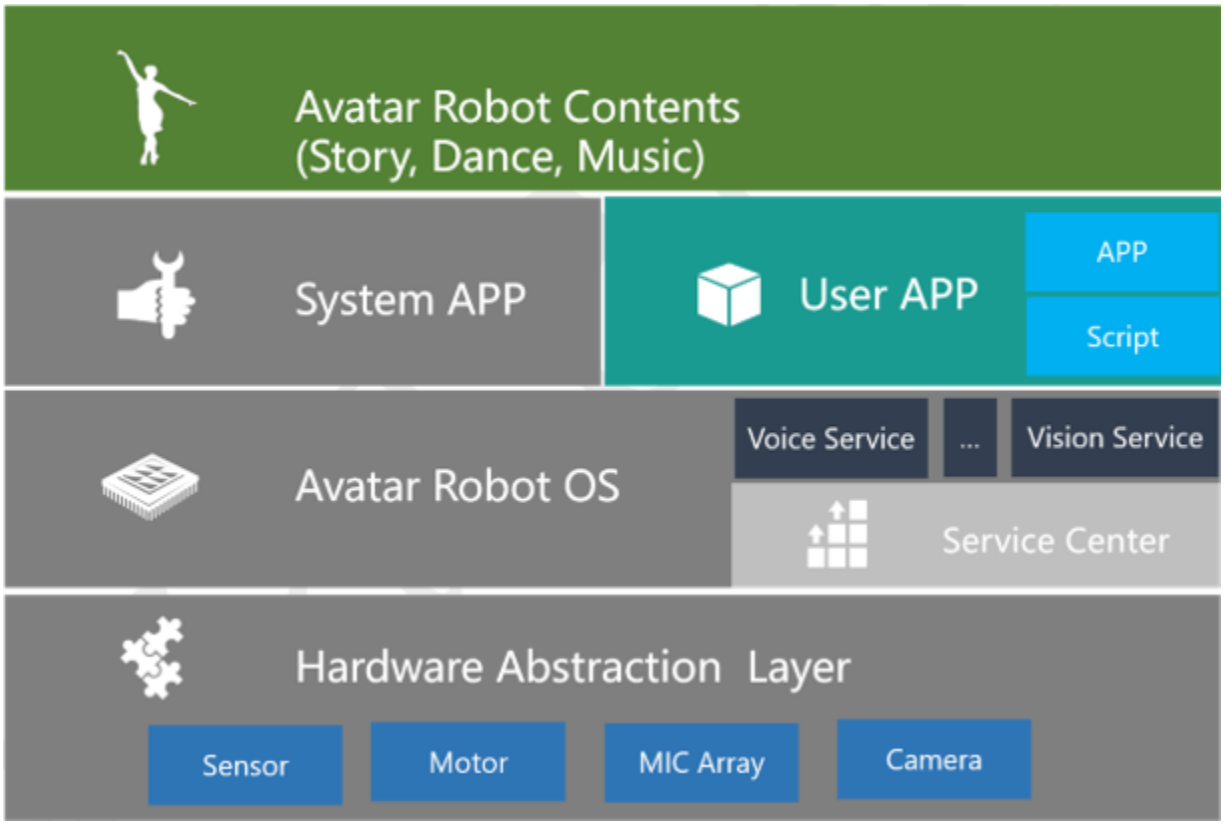
Overview

Introduction

AvatarToolKit is the AvatarMind development platform, based on the AvatarMind robot hardware platform. AvatarToolKit provides third-party developers with a complete robot data interface, a practical development guide, a professional robot simulator, and robust development tools. Along with the industry-leading robot App Store by AvatarMind, AvatarToolKit will assist developers in managing the production cycle of all robot applications and robot content. The app production cycle includes 5 steps for developing a good robot application: development, testing, publishing, management, and operation. AvatarMind will work together with domestic and overseas robot technology providers and developers in order to provide users with the best service robot experience.

Target User

AvatarToolKit is designed for users coming from a wide range of STEM fields including Hardware Manufacturing, System Solutions, Application Development, and Content Design. AvatarToolKit provides flexible and professional solutions for various developers. The structure of AvatarToolKit is shown below:



Hardware Manufacturing(OEM)

AvatarToolKit defines robot hardware specifications and communication protocols. Hardware manufacturers and OEMs may design new robot hardware profiles or update robot device firmware under these requirements.

System Service Provider

AvatarToolKit abstracts system services such as voice recognition into abstract interfaces. Because of this, these services are available for system solutions providers or developers to modify based on their own development requirements.

Application Developer

Developers can design and submit their own applications, provided the applications follow AvatarMind's development guidelines and AvatarMind App Store standards of business conduct. App pricing is determined by the developer.

Content Design

AvatarToolKit provides powerful content editing tools through the AvatarStudio program. Users can record robot motion sequences using a 3-D simulated robot skeleton. By creating these robot motion sequences in **AvatarStudio**, users can design robot dance routines or even robot storytelling routines.

Advantages

The Leading Robot Platform Domestic and Overseas

iPal, AvatarMind's in-house robot, has been invited to multiple world-famous robot exhibitions such as US CES and Asian CES, where it has earned several awards. In field tests, iPal was widely well-received among education institutions and children. Professional robot technology practitioners predict that iPal will have a wide market prospect, because the robot application market has much potential for growth when compared to the saturated mobile application market.

Excellent Customer Service

At AvatarMind, we have senior engineers and experienced customer service to provide assistance for any problems that developers may have. AvatarMind also provides comprehensive documentation to assist users in developing iPal robot applications.

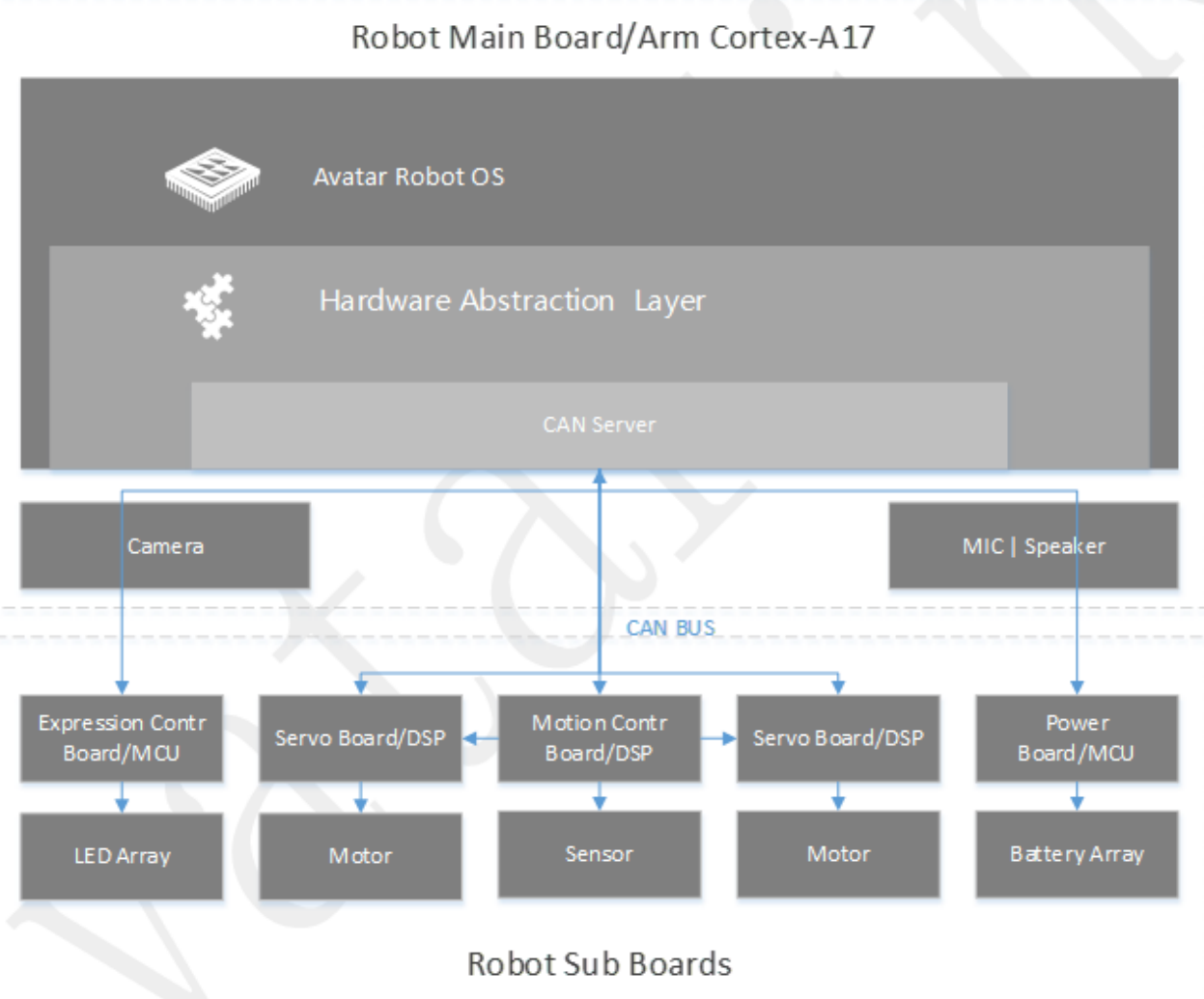
Attractive App Benefits

The application revenue proportion between the developer and Avatarmind is 8:2. AvatarMind also provides additional incentives for developers.

Hardware Design

Hardware Architecture

The robot hardware structure is shown in the diagram below:



Main Board

The main board consists of an ARM board and multiple assisting boards. This component is the core of the robot operating system and takes the role of the robot's brain.

Sub Boards

Sub boards consist of multiple DSP/MCU boards, sensors and motors. These components are responsible for robot device management and robot motion control.

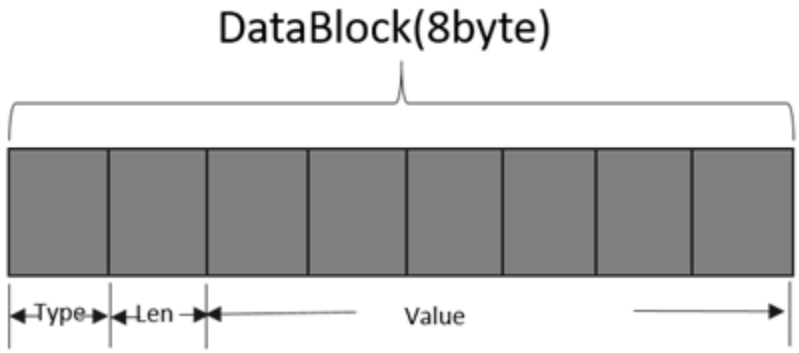
Communication Protocol

CAN Bus Protocol

The CAN Bus is a serial communication protocol bus used for instant communication between components. iPal employs the CAN protocol to quickly transmit data and commands between the main board and sub boards.

TLV Protocol

In iPal's data transmission protocol, info is encoded into the TYPE-LENGTH-VALUE format, i.e. the TLV protocol. All CAN data and command transmissions must follow the TLV protocol. Length of transmitted data is fixed to 8 bytes: TYPE and LENGTH use 1 byte each, and the remaining 6 bytes consist of VALUE. Formal definition of the TLV protocol is shown in the chart below:



Development Model

Customized Devices

AvatarToolKit defines the standard device specification and communication protocol. Developers can add custom sensors and motors or even remove some devices from the system, provided the AvatarToolKit standards are followed.

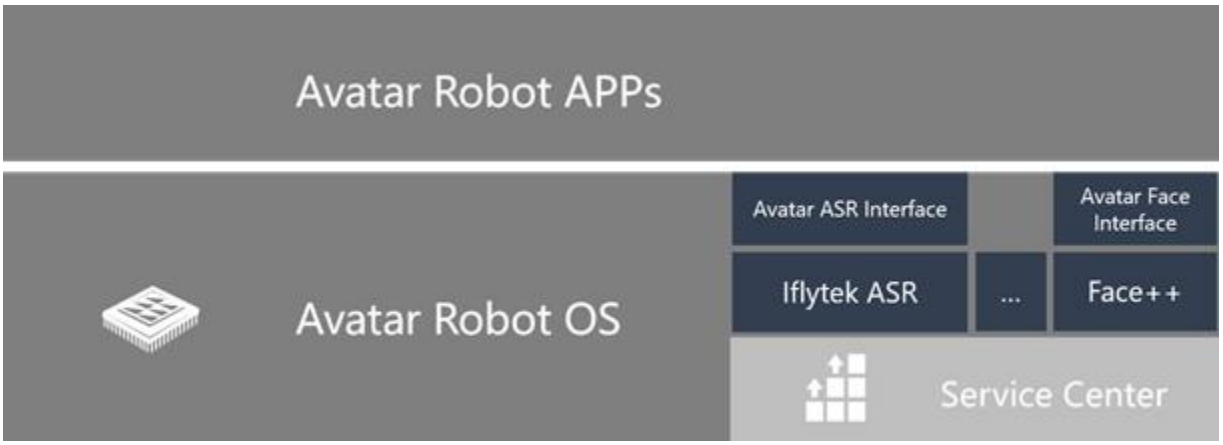
Firmware

AvatarToolKit provides tools for editing and loading firmware to devices. By using these tools, developers can create custom device firmware that matches their target user's needs.

System Services

System Architecture

The system structure of AvatarToolKit is as shown below:



Abstract Service Interface

AvatarToolKit abstracts system services into a standard definition of abstract service interfaces. This design provides the advantage of allowing developers to create custom system services, e.g. a visual identification or voice service. Developers can compare system service plans from various providers, select the best match, and simply rewrite the abstract service interfaces of the desired modules.

Service Management

AvatarToolKit's Service Center is responsible for managing system services. Custom services from developers should strongly adhere to the regulations set by the standard system service. For system security reasons, AvatarToolKit may have some permission limitations on custom services.

Customized Services

Implementing Interface

AvatarToolKit contains the default abstract service interface, which is available for developers to customize. Any abstract interface in AvatarToolKit can be rewritten; for example, if a developer wants to replace the default TTS plan, they can rewrite the TTS abstract interface of the robot system.

System Configuration

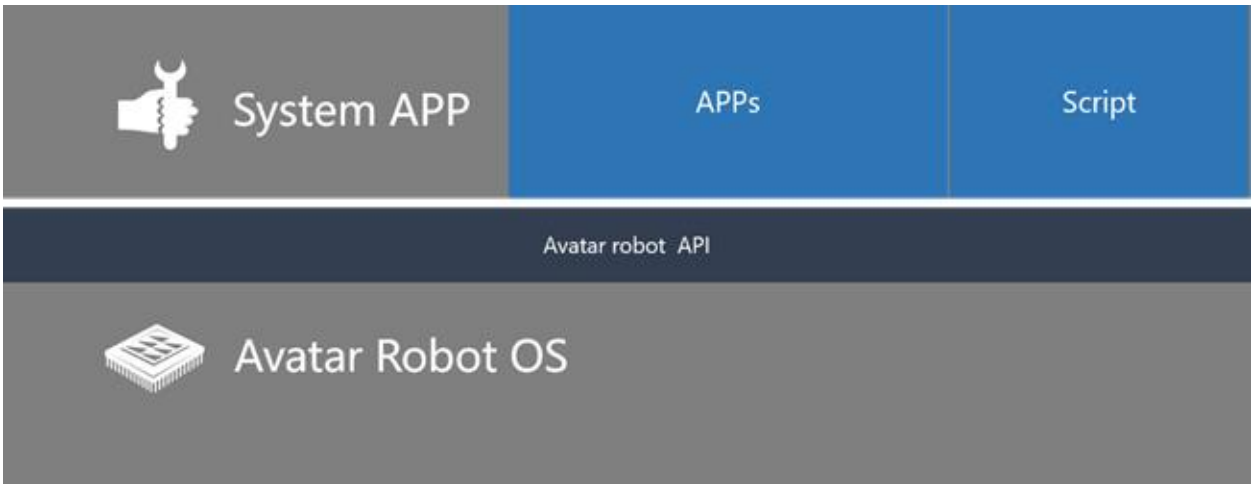
If a developer modifies an abstract service interface, they are required to reconfigure service permissions and other necessary settings.

Registering Services

Developers can call the service registration interface to register new services to the system, replacing default system services.

Application

Application Architecture



APP

An application developed in Java or C/C++, running its own individual process.

Script

JavaScript code which needs to be run through the system script compiler.

Application Development

App Development

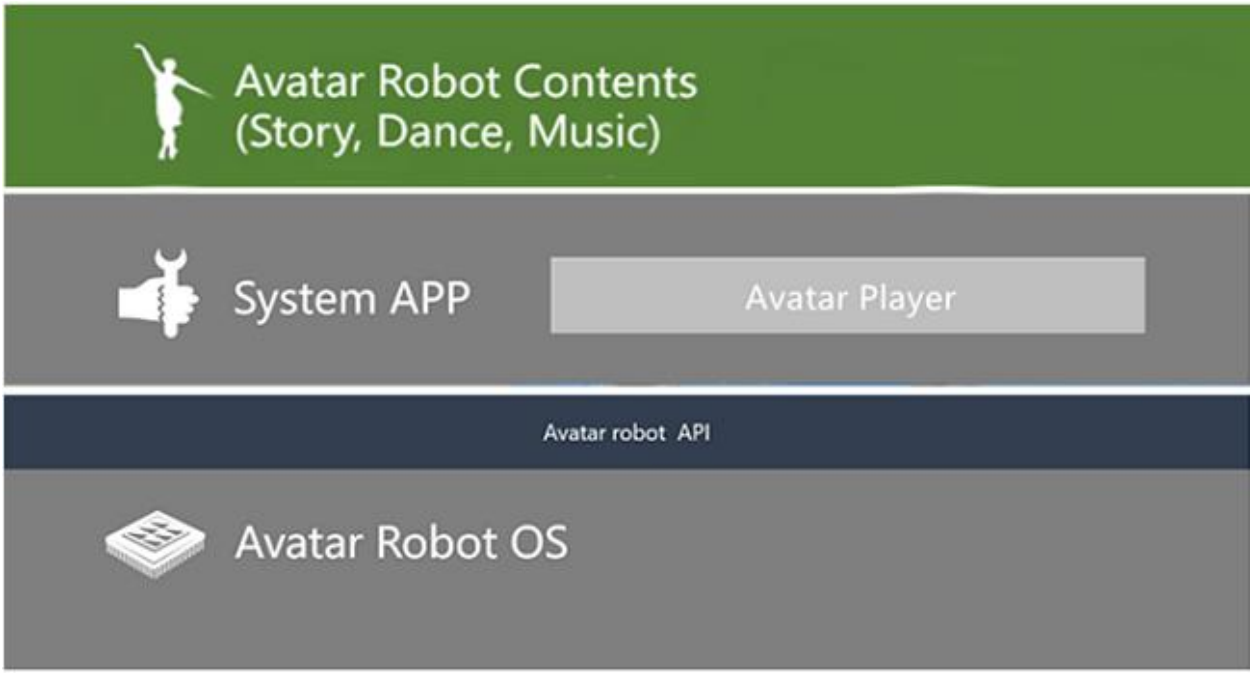
Developers should import the AvatarMind robot SDK for app development. This tool makes it easier for developers to write or debug code. AvatarToolKit also provides users with **AvatarStudio**, a 3-D robot simulator. Developers should have at least an intermediate level of Java/C/C++ developing experience. AvatarToolKit provides the user with a series of development guides and samples to guide developers on using the AvatarMind SDK.

Script Development

Scripts can be easily developed using the Robot Programmer tool in AvatarToolKit. Robot Programmer provides an environment in which code is in the form of blocks. This type of simplified programming makes it easier for developers to create simple logic for robot applications.

Creating Content

Content Architecture



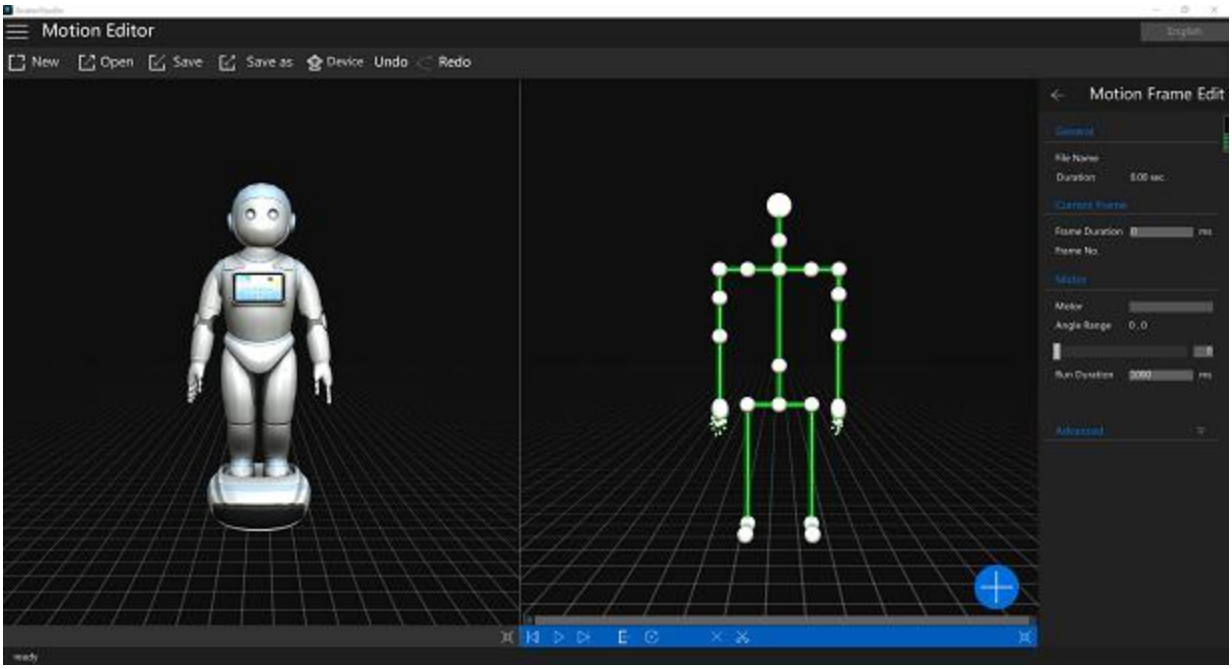
Content Introduction

- Robot Motion File(.ARM)
  The .ARM file extension is a motion file type defined by AvatarMind which can be processed on the robot. A single .ARM file consists of multiple motion frames, and each frame contains angle info for all robot motors.

- Robot Content File(.ARC)
  .ARC files consist of an .ARM file, media resources (video or music), subtitles and script.

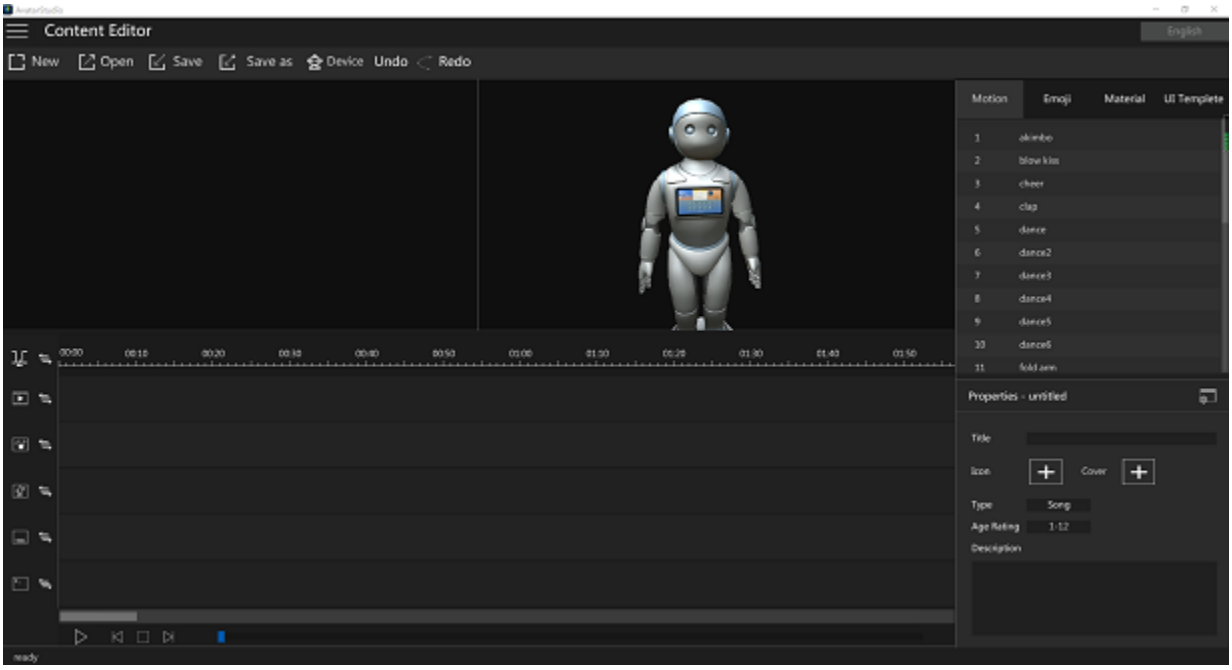
Motion Editor

Users can create or edit motion resources by motion editor from AvatarToolKit, as shown below:
Motion resources are saved as ARM files. ARM files can be either operated on the editor simulator or uploaded to robot.



Content Editor

AvatarToolKit provides a content editing tool which is used to create or edit robot content as, shown below:
After an ARC file is completed, users can upload this file to a robot or to the AvatarMind App Store.



Dimensions and Weight

- Height: 103 cm/3.5 feet
- Weight: 12.5 kg/27.5 pounds

Software

- Android OS
- Full SDK
- Content & Motion Editor. Easily create content and new activities - drag and drop along a timeline
- Remote control with telepresence
- Most standard Android apps can be run on the chest mounted display

Connectivity

- WiFi
- Bluetooth
- Full cloud connectivity

CPU/Memory

- Rockchip 3399
- 1.8GHz Quad core
- Mali-T860 GPU
- 4GB RAM
- 32GB ROM
- 7 MCUs and 3 DSPs at various locations to control motors, sensors, etc.

Motors

- The 14 12V DC motors have magnetic encoders to measure angles
- 5 motors in each arm -- two at shoulder, two at elbow, and one at wrist
- 2 motors in the neck to move the head side-to-side and up-down
- 2 motors in the base for locomotion

Batteries

- 2 options
- Robot Basic version comes equipped with a small battery that gives approximately 4 hours of continuous usage
- Robot Professional version comes equipped with a large battery that give approximately 8-10 hours of continuous usage

Standard Software

- Conversational speech dialog, natural language understanding, text to speech
- Face recognition, object tracking and following, maze-running
- Obstacle collision avoidance
- Remote Control and safety monitoring by smart phone, telepresence
- Numerous entertainment applications (Songs, Stories, Dances, etc.) and educational applications (for teaching English, Math, Science, Technology, etc.)
- Software to manage, update and enhance content
- iPalProgrammer for programming robot with a simple drag and drop interface
- High level content editor to enable non-programmers to develop robot content by combining media (such as a song), robot motions and expressions etc. together.
- Emotion Recognition & Response

Microphones

- 1 in chest

Sensors

- 3 infrared sensors for short range object detection
- 5 ultrasound sensors for long range object detection
- 5 touch sensors on surface

Chest Display

- 10.1 inch screen

Miscellaneous

- iPal App Store for new apps, education and entertainment content, upgrades and related products
- Robust modular design, easily repairable parts
- Assembly line designed for production of thousands of units per month
- Different color highlights and other customizations are readily available
- No gaps in robot that can catch or pinch fingers
- Batteries installed in the base to lower the center of gravity and make tipping over unlikely
- Sensors are on at all times, so robot avoids obstacles and moves only when safe
- Outer surface made of non-toxic ABS
- Tested for collisions, impacts, etc

Overview

This document will guide you in setting up the AvatarMind SDK in Android Studio, as well in developing robot app projects in Android Studio.

Hardware Profile

Recommended:

- CPU: 3.40GHz dual-core or higher
- Resolution: 1920x1080
- RAM: 8G
- Ubuntu: 14.04 LTS or higher

Environment Setup

JDK

JDK version must be at least 1.6.

Download SDK

Please download and unzip SDK for your OS:

Windows
Linux

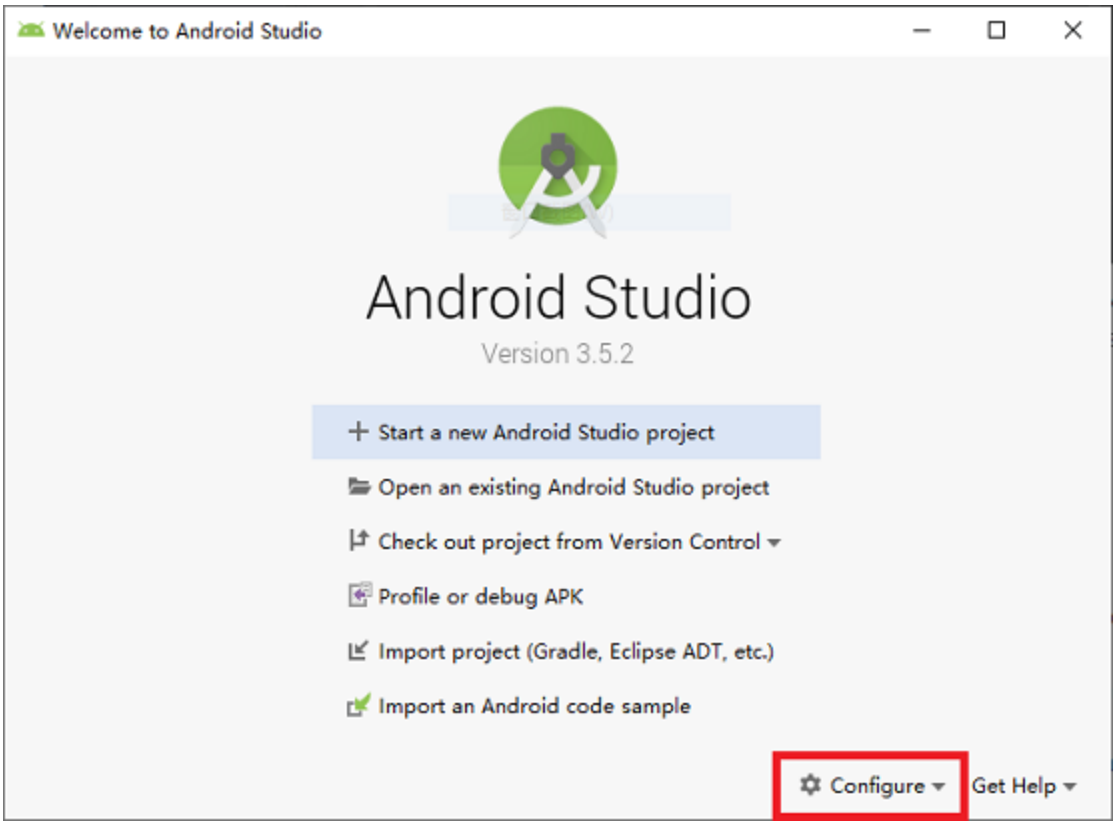
Download Android Studio

Please download and unzip Android Studio for your OS:

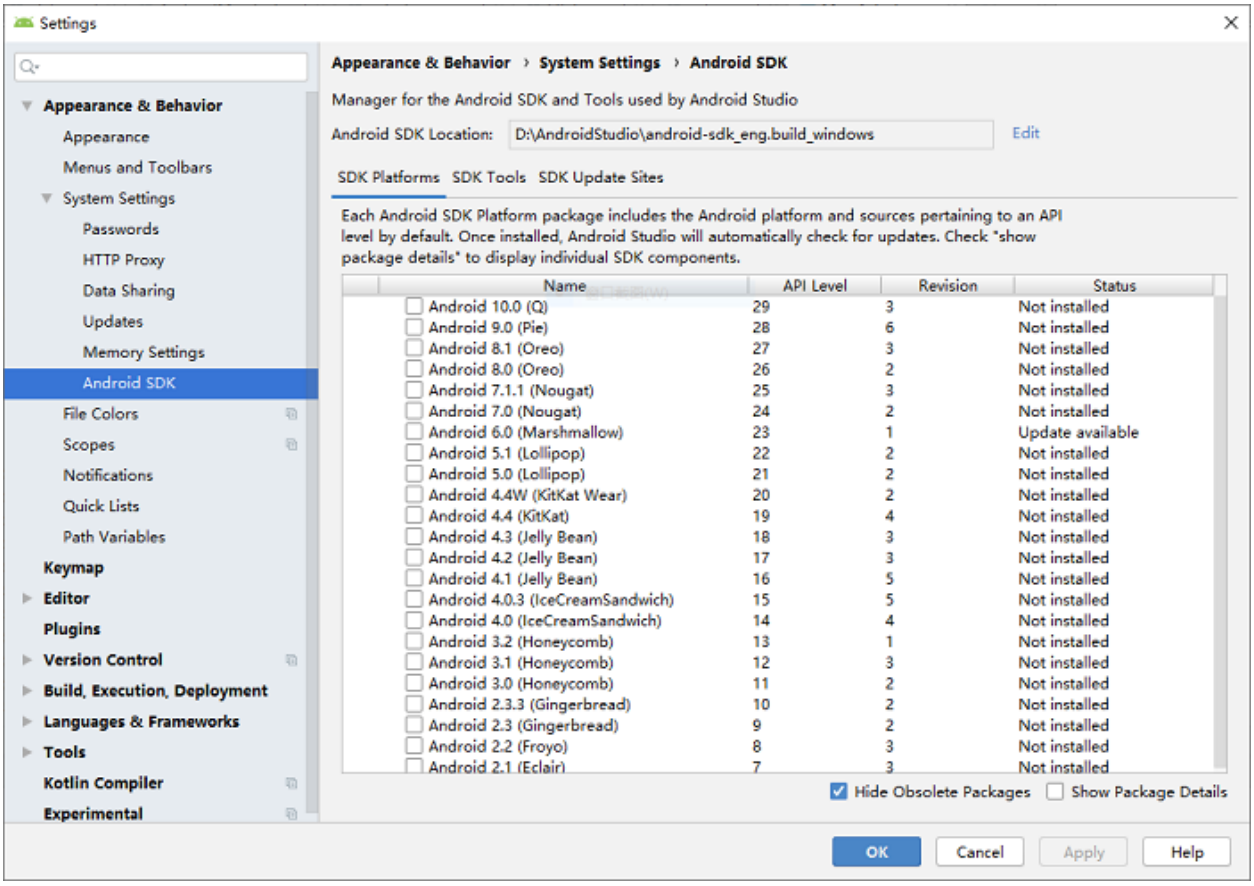
Windows
Linux

Setup SDK

- Enter to the Welcome page of Android Studio. Navigate to the "Configure" icon -> Settings.



- Enter to Settings -> System Settings, then select [Android SDK] and edit Android SDK Location to your AvatarMind SDK folder path. The page is shown below:



Note: While the AvatarMind Robot OS is based on the Android 7.1 platform, AvatarMind SDK contains extensive modifications to the Android SDK. Android Studio may ask you to update your SDK version. **Do not update the SDK.** If updated, unexpected errors may occur.

Since our SDK is compiled based on Android 7.1, the compiled SDK toolkit is relatively old and prone to problems that are not compatible with the new version of AndroidStudio.

Here's a way to replace the platforms in the native SDK:

1. has AndroidStudio and a set of native Android sdk, can compile native Android applications
2. download avatar sdk, unzip after downloading (assuming the directory name is avatar-sdk), the directory structure is similar to the native Android sdk (assuming the directory name is android-sdk)
3. Make sure the path of avatar sdk your input is correct, cause that the avatar sdk when you unzip it, it will be saved in a sub-directory. If not, when you direct Android Studio to the sdk directory it will not recognize that the SDK is in the sub-directory so it will start to install only the tools and cannot find the right sources from the SDK.
4. delete the android-sdk/platforms/android-19 and android-sdk/platforms/android-25 two directories in the native Android sdk, if not, do not deal with
5. copy the avatar-sdk/platforms/android-7.1.2 directory (7.1 version avatar-sdk) to the android-sdk/platforms directory
6. open the Avatar application in AndroidStudio, use the original android-sdk directory sdk, change the application compileSdkVersion, targetSdkVersion to 25 **(although it should be automatically authorized to version 25, it also has to be changed to 25 manually by yourself)**, and change the minSdkVersion to 19

7. Gradle sync application, if there is an error, it is because the new version of AndroidStudio automatically added some content that is not in android-25, can be deleted

Note: This method of use can not use the virtual machine to debug the application, only installed in the robot.

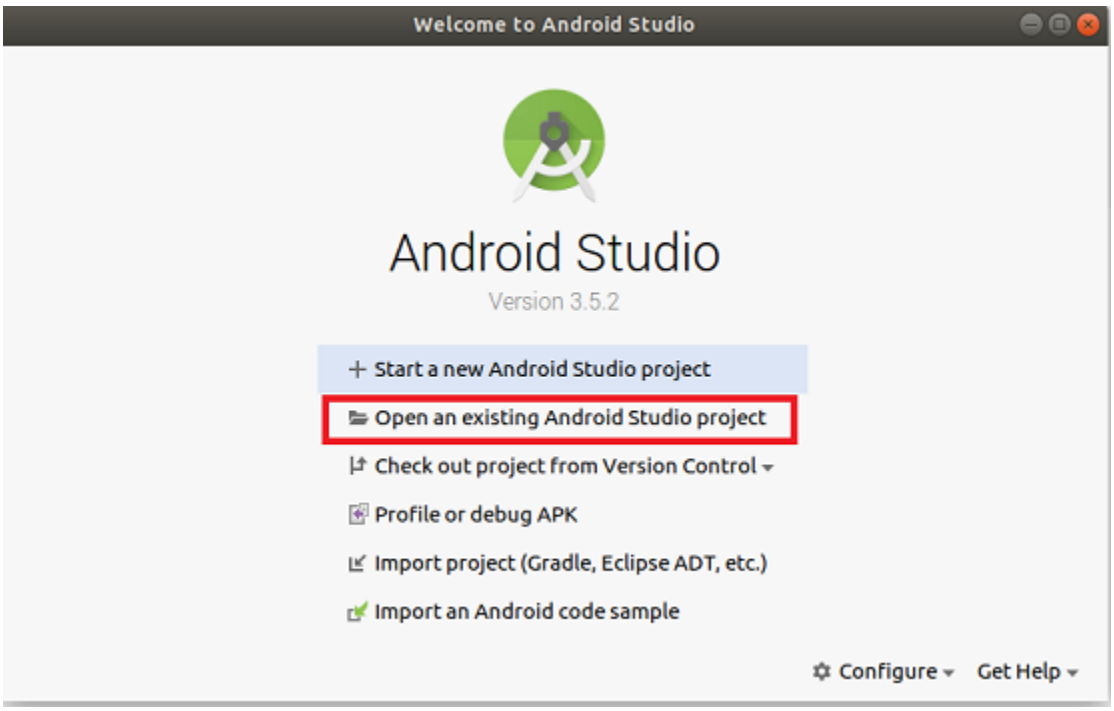
Hello World

Introduction

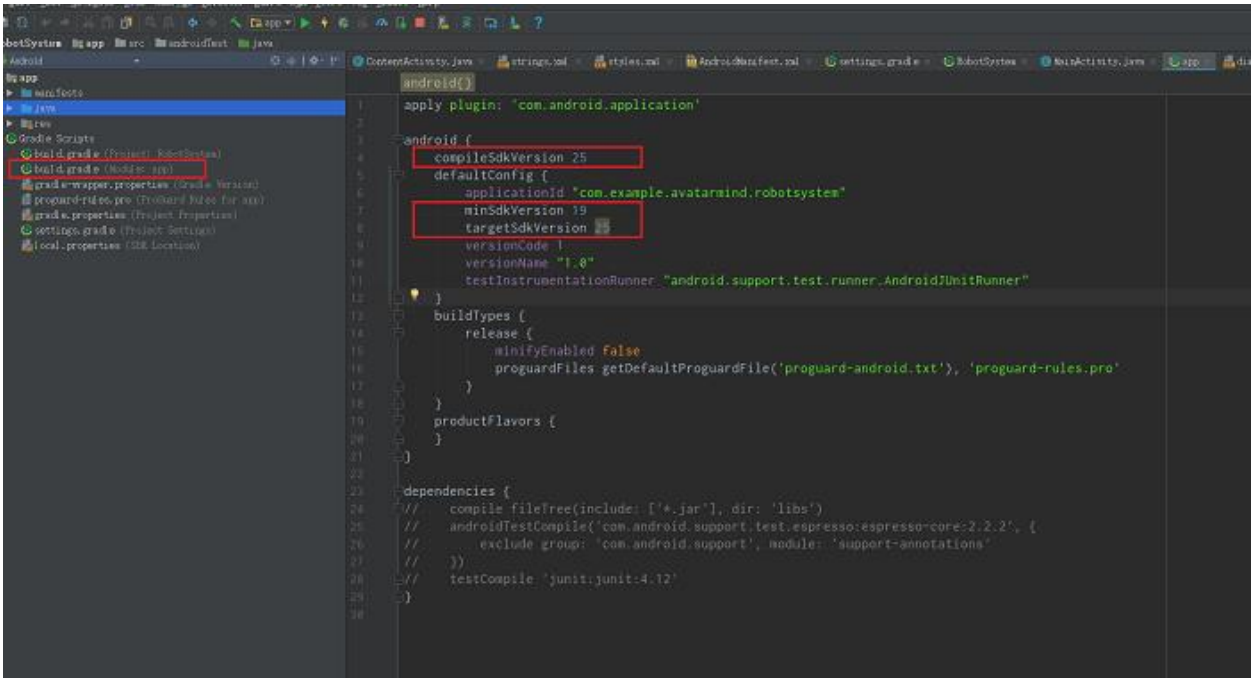
This chapter will introduce developing your first iPal application in Android Studio. Before creating a new project, please make sure that Android and the AvatarMind SDK are properly set up. It is strongly recommended to develop with an iPal robot, as it will make it easier to design and debug code.

Open Demo Project

- Download as well as unzip the *Demo project*.
- Enter to the Welcome page of Android Studio. Click "open an existing Android Studio project" and select the Demo project you download.



- After the Demo project is imported, you must ensure the application compileSdkVersion, targetSdkVersion are 25, and the minSdkVersion is 19.
  Please navigate to Android -> Gradle Scripts -> build.gradle(Module:app) to check those three SDK versions.



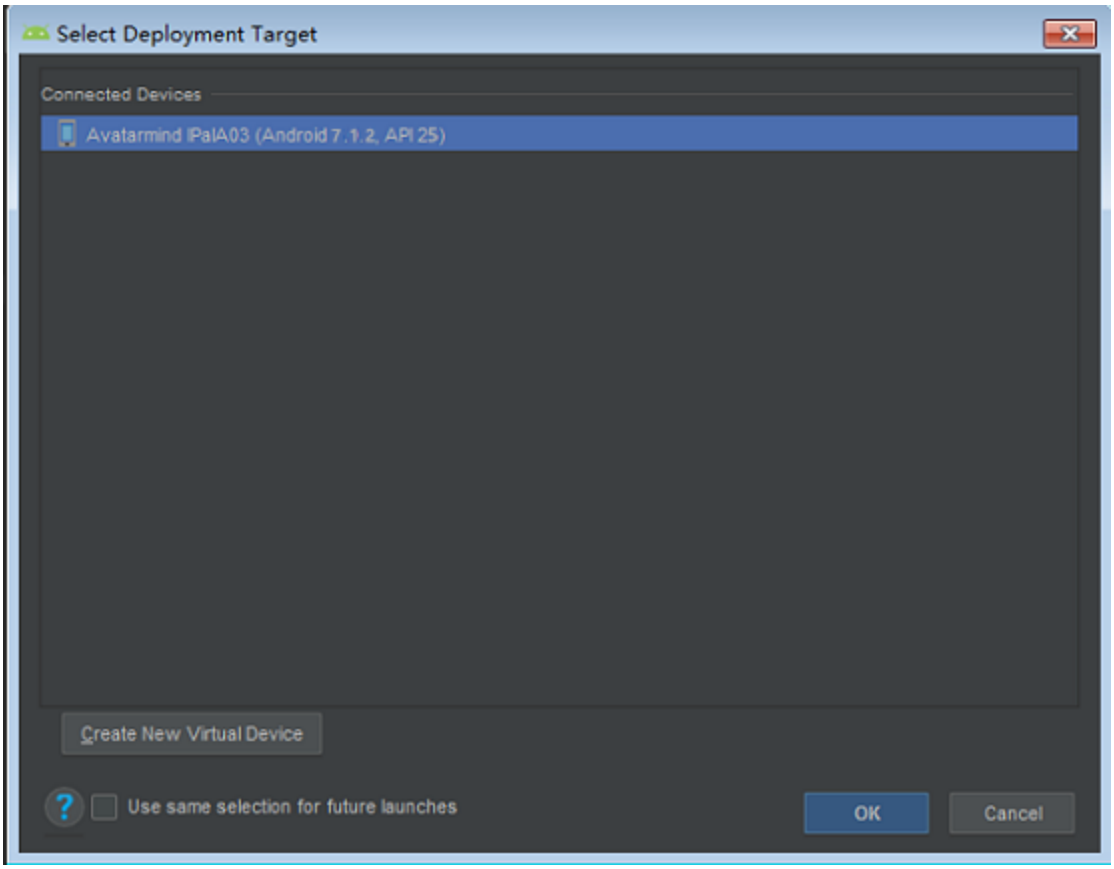
- After checking the SDK versions, you can begin to write code in this project. Sample code is shown below:

MainActivity.java

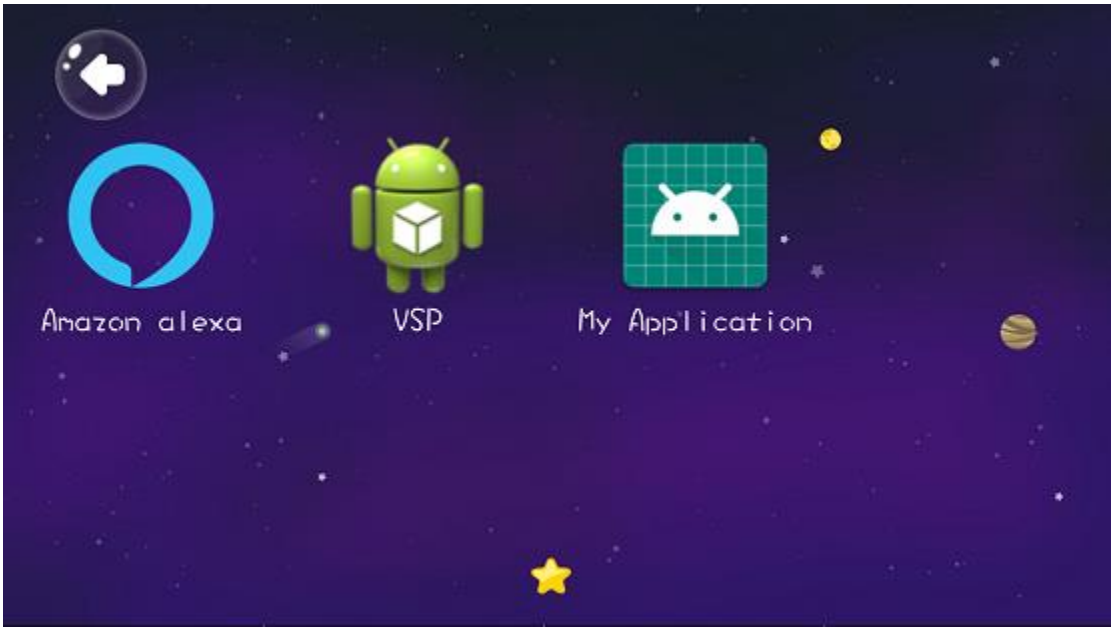
```java
package com.avatarmind.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.robot.speech.SpeechManager;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity implements View.OnClickListener{

    private SpeechManager mSpeechManager;

    private Button mBtnBack;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mSpeechManager = (SpeechManager) this.getSystemService( name: "speech");
        mSpeechManager.forceStartSpeaking( text: "Hello World");

        mBtnBack = (Button) findViewById(R.id.btn_left);
        mBtnBack.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btn_left:
                finish();
                break;
        }

    }
}
```

Debug

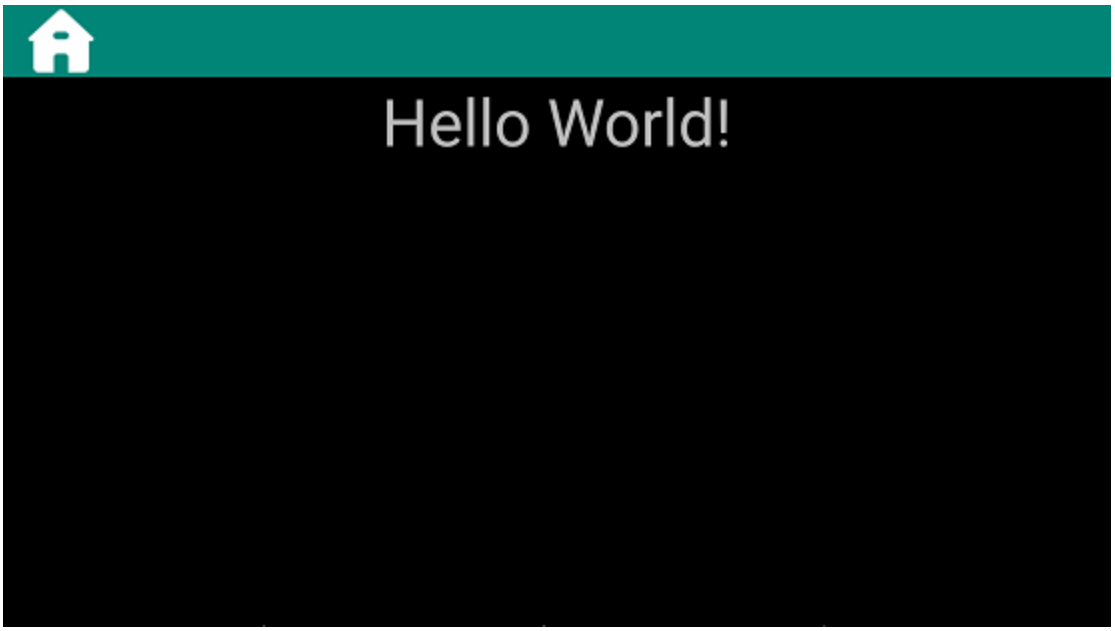
- To debug a project on an iPal robot, connect the robot to your PC via USB:



- Install the sample project to your iPal. The default install path is the [Others] folder:



- Open My Application and your iPal will show "Hello World!" on screen and also say "Hello World" to you.



Note: Android Studio version recommended **3.5**, AndroidStudio may prompt to upgrade, **please do not upgrade**, you can open the project directly

Overview

This document will introduce how to use the RobotMotion class to make your robot do specified actions. RobotMotion actions can be divided into the categories listed below:

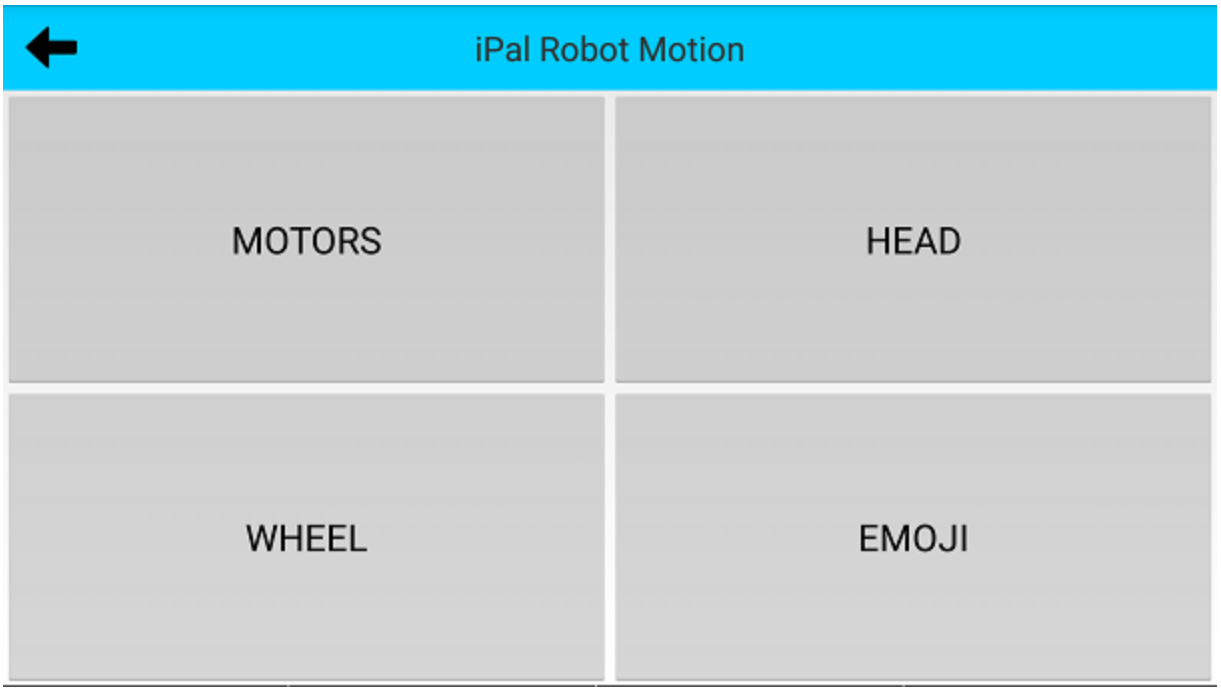
- Motor
- Head
- Wheel
- Expression

In the sample below, we will create an Android Studio project to demonstrate the functionality of RobotMotion.

Sample Introduction

Create New Project

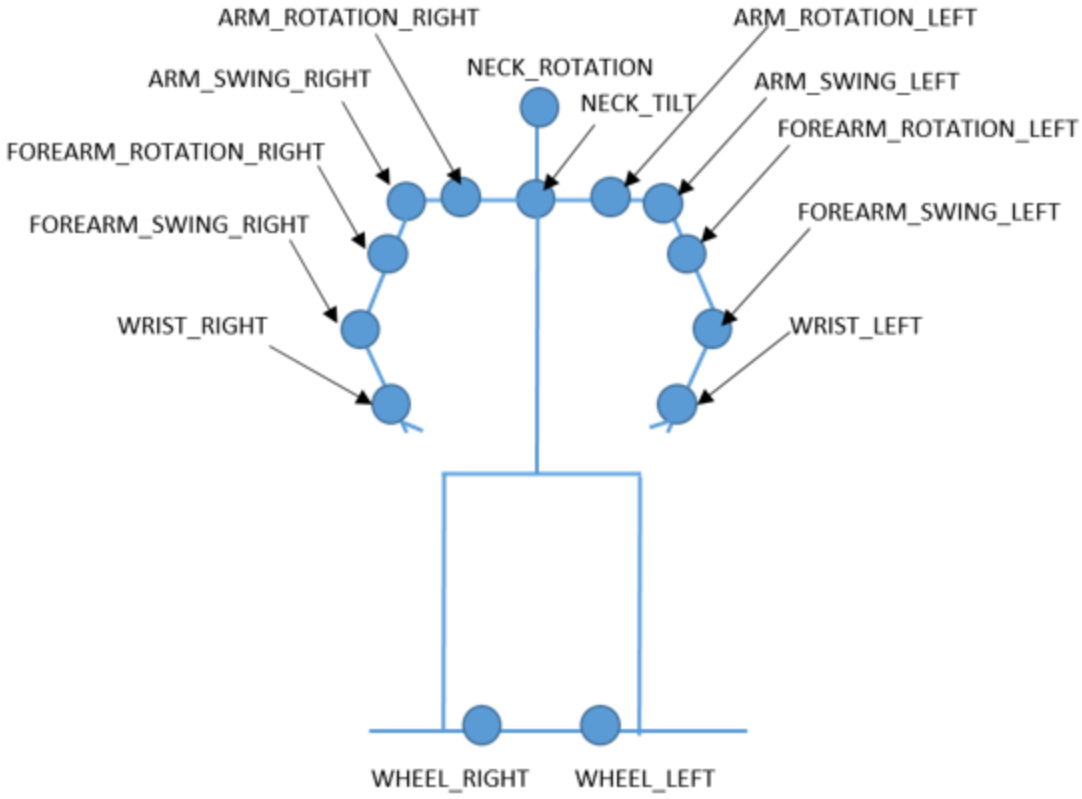
Before creating a new robot application project, make sure that Android Studio and the AvatarMind SDK are properly set up. Create a new project and import the RobotMotion package: android.robot.motion. The main activity of the sample project is shown below:



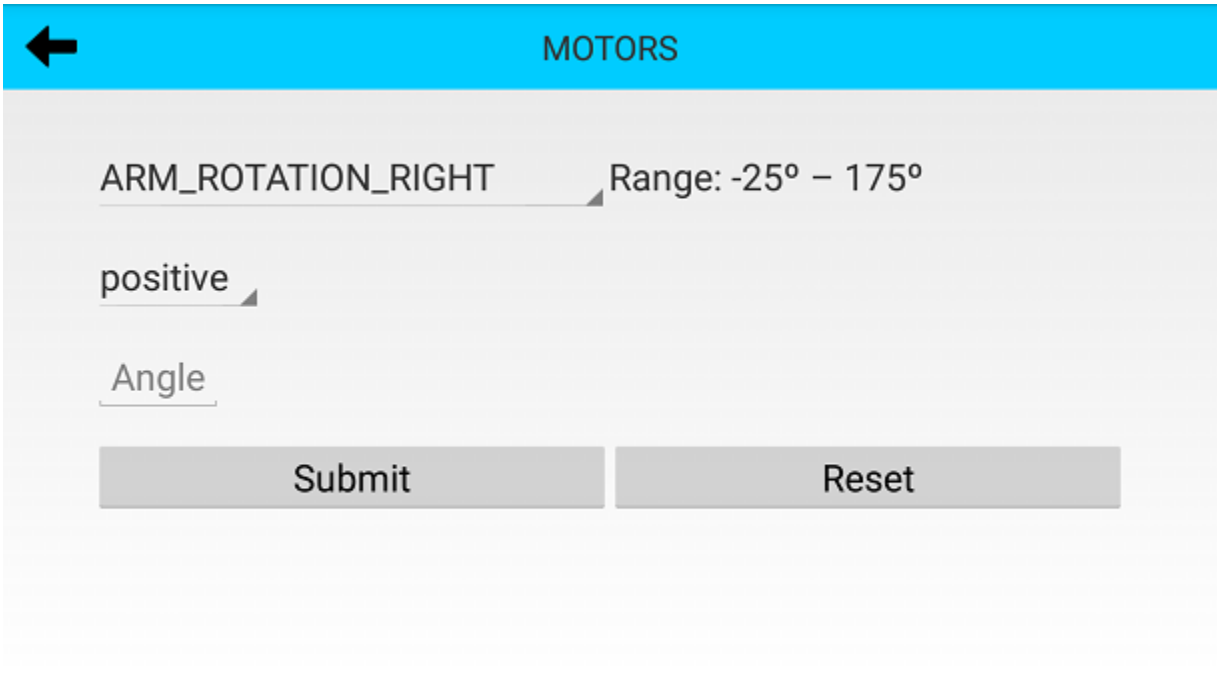
- MOTORS: Shows controls for the robot's body motors
- HEAD: Shows controls for the robot's head
- WHEEL: Shows controls for the robot's wheels to move or turn
- EMOJI: Shows controls for the expression panel

Joint Control

All moveable joints are shown below:



The Motor control activity is shown below:



Operation

1. Select a motor

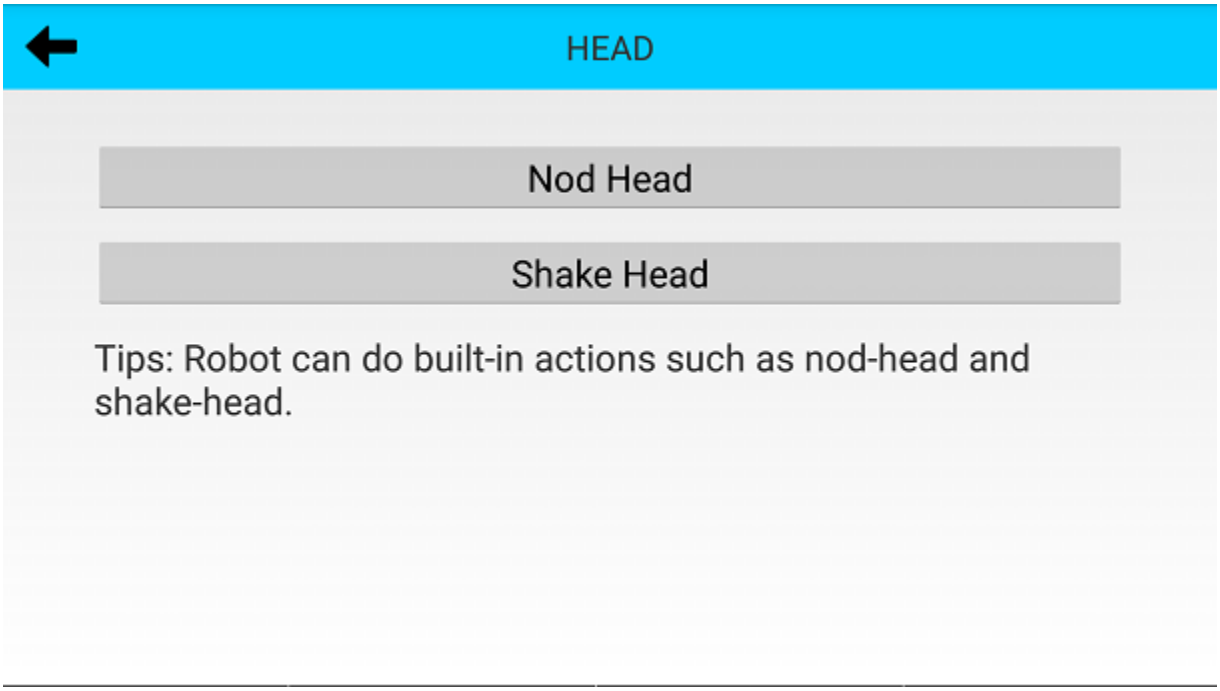
2. Select direction
3. Enter an angle (within the range allowed for the motor)
4. Submit: Run command
5. Reset: Reset all body parts

Sample Code

```
// run motor
RobotMotion mRobotMotion = new RobotMotion();
int action = mAction.getSelectedItemPosition();      // get which motor is going to be operated
int direction = mDirection.getSelectedItemPosition();    // get direction
String degree = mAngle.getText().toString();        // get angle
int angle = Integer.parseInt(degree);

if (direction == 1) {
  angle = angle - 2 * angle;
}
switch (action) {
  case 0:
      mRobotMotion.startMotor(RobotDevices.Motors.ARM_ROTATION_RIGHT, angle, 1000, 1);       //
this motor will rotate assigned angle in 1 second
      break;
  //...
}

// reset motor
mRobotMotion.reset((int) RobotDevices.Units.ALL_MOTORS);
```

Head

The Head control activity is shown below:

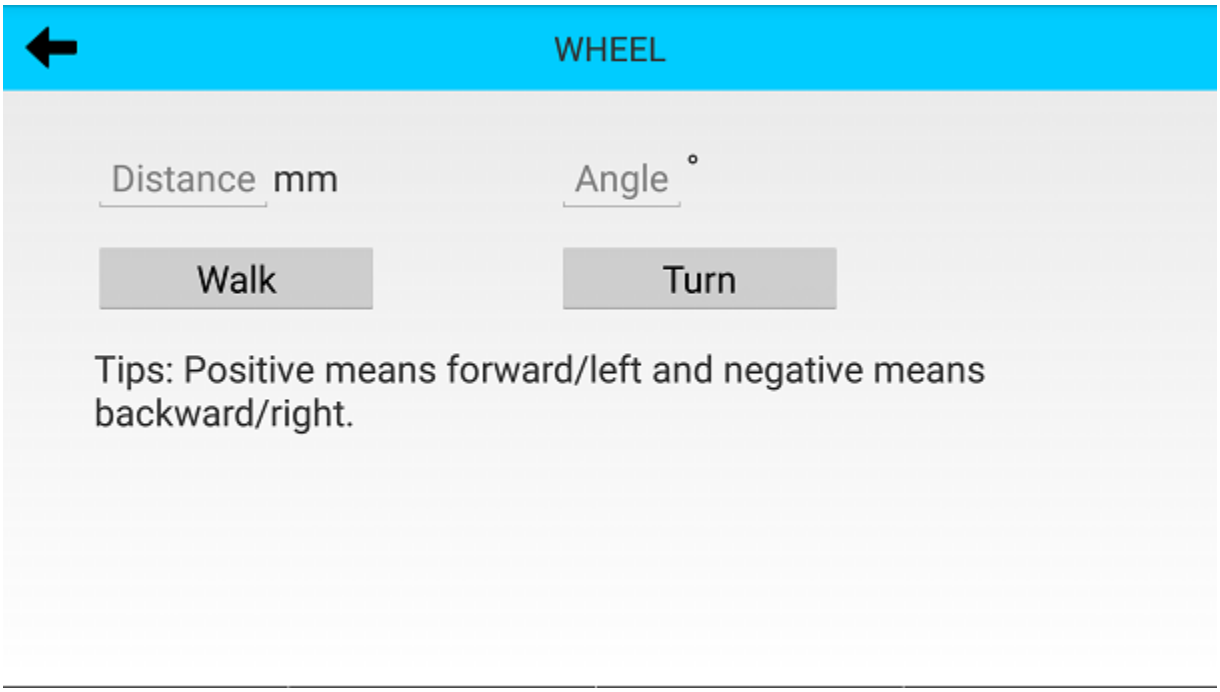


Operation

1. Nod Head: iPal nods its head
2. Shake Head: iPal shakes its head

Wheel Control

The Wheel control activity is shown below:



Operation

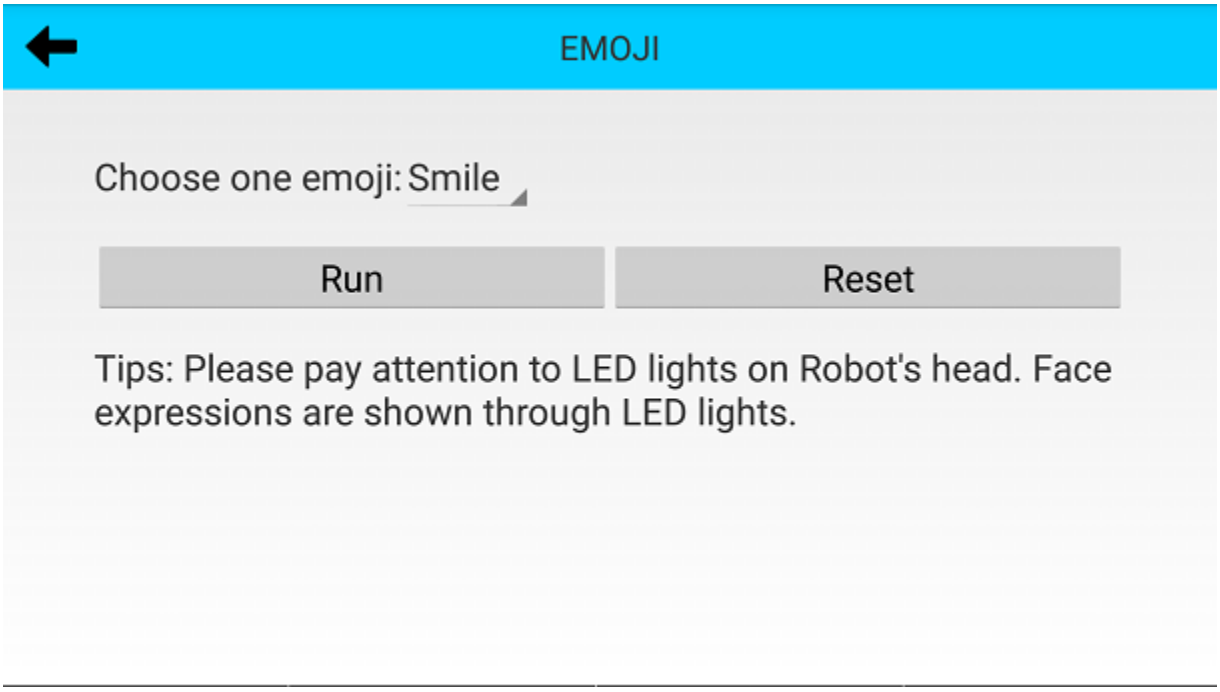
1. Enter a distance in millimeters, positive is forward and negative is backward
2. Walk: Walk the specified amount
3. Enter an angle to turn, positive is left and negative is right
4. Turn: Turn the specified amount

Sample Code

```
// walk
String length = mEtDistance.getText().toString();
if (!TextUtils.isEmpty(length)) {
  int distance = Integer.parseInt(length);
  mRobotMotion.startWalk(distance, 1, 0);
}

// turn
String degree = mEtAngle.getText().toString();
if (!TextUtils.isEmpty(degree)) {
  int angle = Integer.parseInt(degree);
  mRobotMotion.turn(angle, 2);
}
```

Expression Control

The Expression control activity is shown below:



Operation

1. Select a robot expression
2. Run: Display the expression
3. Reset: Reset to default expression

Sample Code

```
// run expression
int position = mEmojiSelector.getSelectedItemPosition();    // get expression
mRobotMotion.emoji((int) EMOJI_ARRAY[position]);

// reset expression panel
mRobotMotion.emoji(RobotMotion.Emoji.DEFAULT);
```

Full Source

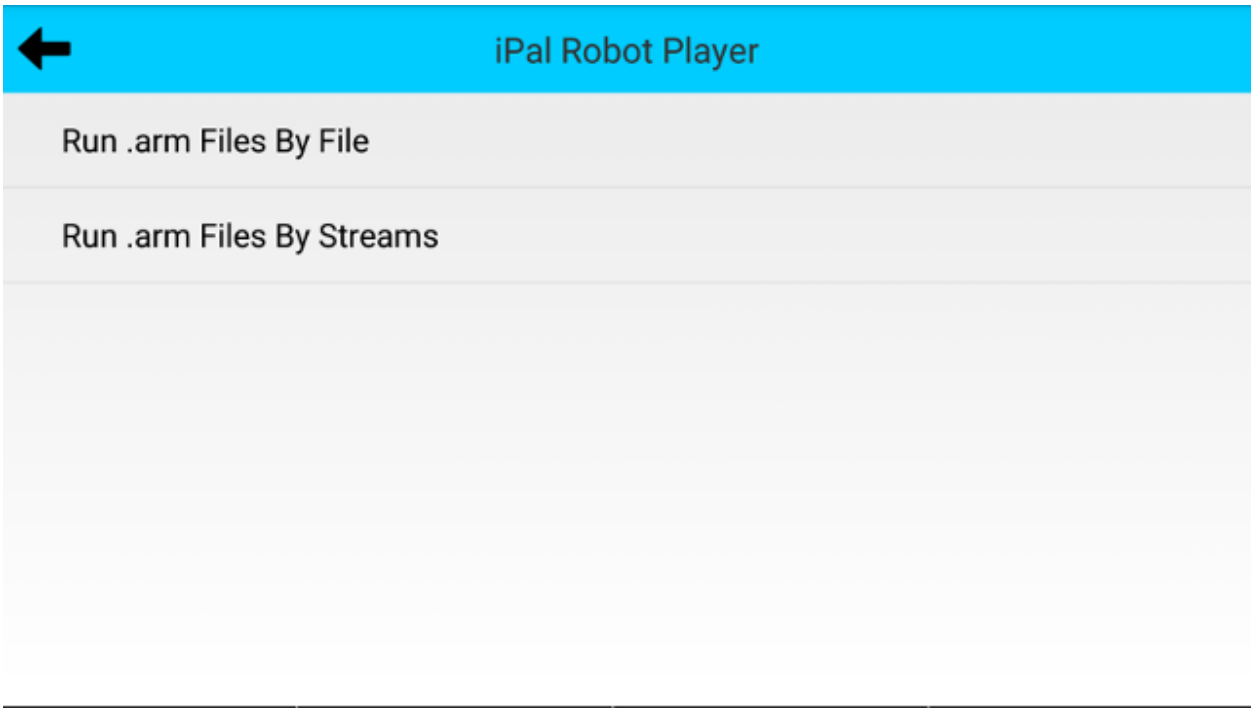
The full source for this tutorial is available **here**.

Overview

This document will introduce the RobotPlayer class and provide sample code. The RobotPlayer class is used to play ARM files (Avatar Robot Motion), which describe AvatarMind robot movements. RobotPlayer provides methods to play local ARM files and ARM-format byte streams.

Sample Introduction

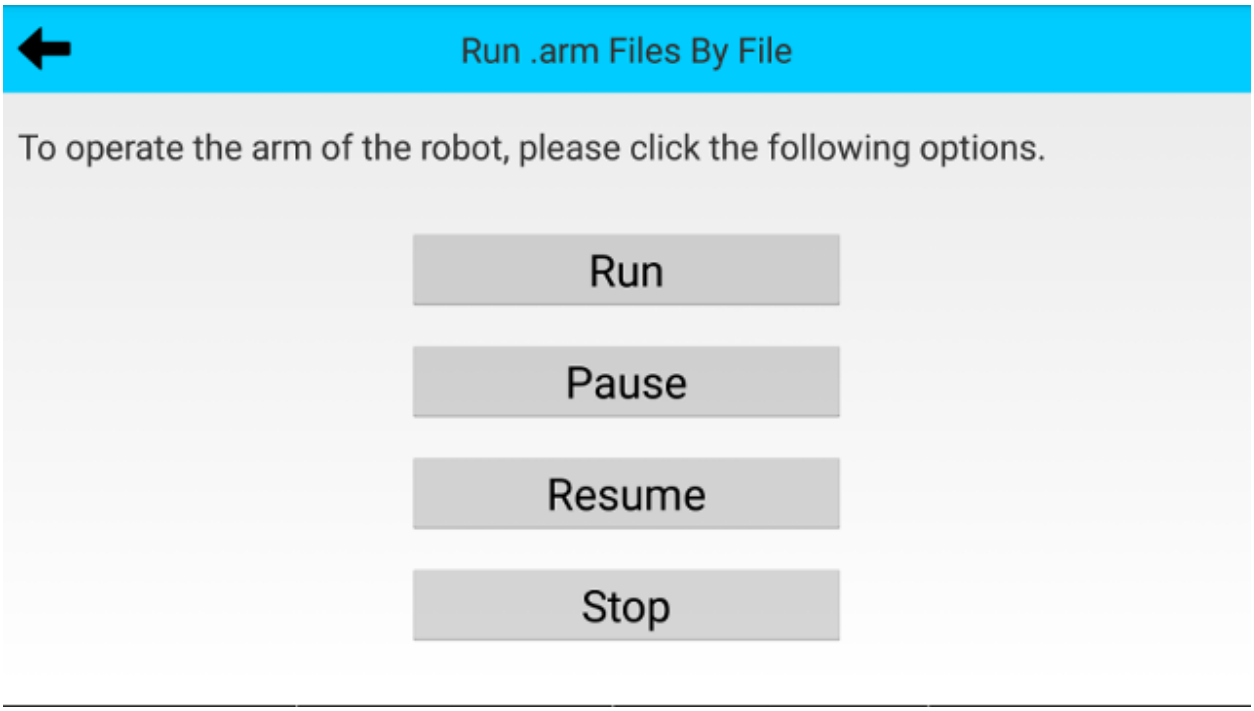
Before creating a new robot application project, please make sure that Android Studio and the AvatarMind SDK are properly set up. Create a new project and import the RobotPlayer package: android.robot.motion.RobotPlayer. The sample project home page is as shown below:



- RunArmByFile: Play local ARM files
- RunArmByStreams: Play ARM-format byte stream

ARM File

The screen to play local ARM files from is as shown below:



- Run: Start playing the ARM file
- Pause: Pause the ARM file
- Resume: Resume the current ARM file
- Stop: Stop the current ARM file. You cannot resume once you have stopped the file

Note: The default ARM file path is /sdcard/media/test.arm. This is located within the robot's storage.

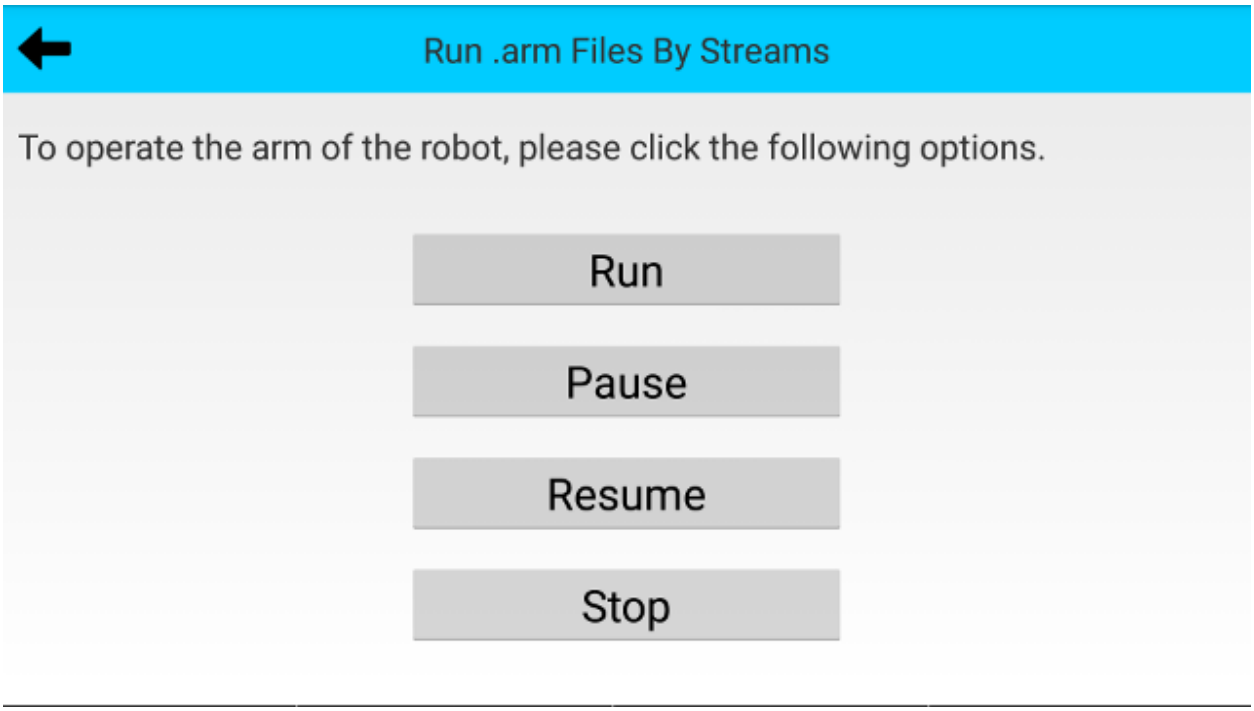
This playing type requires that the ARM file be uploaded to the robot.

Sample Code

```
// Load local ARM file
RobotPlayer mRobotPlayer  = new RobotPlayer();
mRobotPlayer.setDataSource("/sdcard/media/test.arm");
mRobotPlayer.prepare();

// Start
mRobotPlayer.start();
// Pause
mRobotPlayer.pause();
// Resume
mRobotPlayer.resume();
// Stop
mRobotPlayer.stop();
```

Binary ARM Data Stream

The screen to play ARM binary streams as shown below:

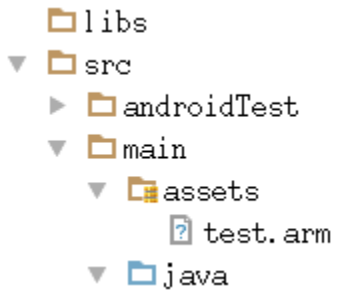


- Run: Start playing the ARM file
- Pause: Pause the ARM file
- Resume: Resume the current ARM file
- Stop: Stop the current ARM file. You cannot resume once you have stopped the file

An example ARM file path is /7.1/assets//test.arm. This file path is in your Android project.

Files do not need to be uploaded to the robot when using this method. You can package the ARM files within the Android Project

The project structure should be as shown below:



Sample Code

```
// Load binary ARM data stream
int mArmLen = 0;
private byte[] getFrom7.1/assets/(String fileName) {
  try {
    InputStream in = getResources().get7.1/assets/().open(fileName);
    // get file's byte count
    mArmLen = in.available();
    // new byte array
```

```
    byte[] buffer = new byte[mArmLen];
    in.read(buffer);
    return buffer;
  } catch (Exception e) {
        e.printStackTrace();
    }
  return new byte[0];
}

// play ARM frame sequence
RobotPlayer mRobotPlayer = new RobotPlayer();
mRobotPlayer.setDataSource(getFrom7.1/assets/("test.arm"), 0, mArmLen);
mRobotPlayer.prepare();

// Start
mRobotPlayer.start();
// Pause
mRobotPlayer.pause();
// Resume
mRobotPlayer.resume();
// Stop
mRobotPlayer.stop();
```

Full Source

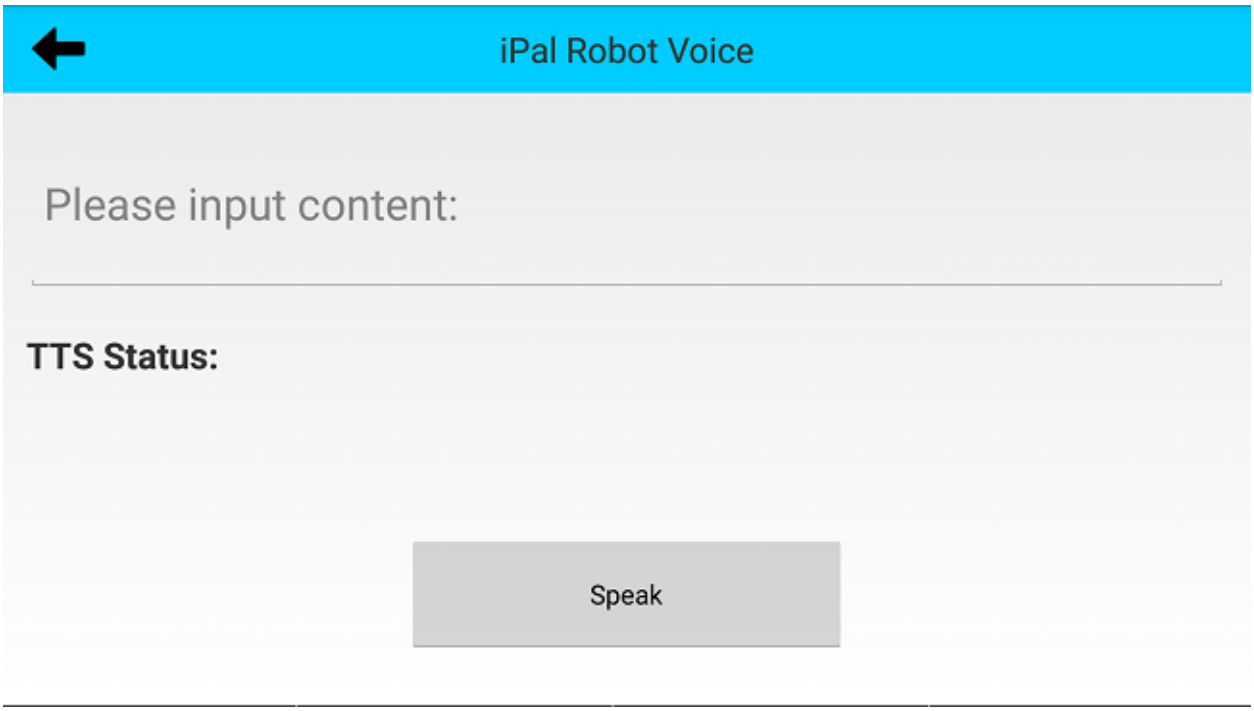
The full source for this tutorial is available **here**.

Overview

This document will introduce the SpeechManager class and provide sample code. Uses for SpeechManager:

- Call the Text-to-Speech (TTS) interface

Home Page



On the page is control for the TTS module.

- **Speak**: The text entered above will be processed through the TTS
- **TTS Status**: Show results from TtsListener

Modules

TTS

Convert text into speech

```
// TTS will convert text to speech
// Output is an id which is used in TtsListner
mSpeechManager.startSpeaking(tts);
```

Listener for TTS

```
// Define a TtsListener
private TtsListener mTtsListener = new TtsListener() {
    @Override
    public void onBegin(int requestId) {
        mTTSStatus.setText(getString(R.string.tts_start_speaking) + requestId);
    }

    @Override
    public void onEnd(int requestId) {
        // User can identify which TTS command is finished through requestId
        mTTSStatus.setText(getString(R.string.tts_stop_speaking) + requestId);
    }

    @Override
```

```
    public void onError(int error) {

    }
};

// Register TTS listener into SpeechManager
mSpeechManager.setTtsListener(mTtsListener);
```

Notes

Destroying Listeners

Set the listeners to null when this API is not in use

```
mSpeechManager.setTtsListener(null);
```

stopSpeaking

You can call `stopSpeaking(int id)` to end a speech process. If the input id is -1, TTS will stop speaking the current text and will clear all speaking commands in the buffer.

onEnd

The TTS Callback onEnd may not always be triggered. For example, the robot may receive a command to speak "Hello" and a return id of 20. However, if the robot is currently speaking text with an id of 19, and stopSpeaking(-1) is called, the original call with id 20 is cleared from the buffer and will not receive an onEnd callback. Therefore it is necessary to add a protection mechanism if your code relies on the onEnd callback.

Full Source

The full source for this tutorial is available **here**.

Overview

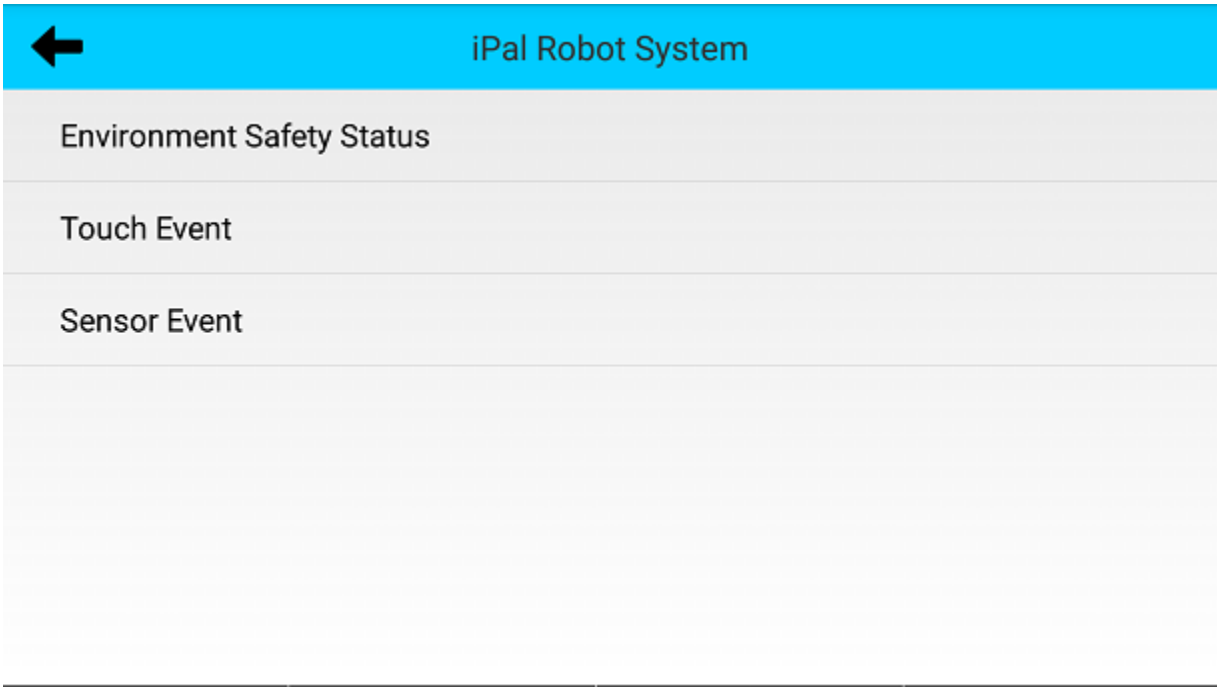
This document will introduce the RobotSystem class and provide sample code. The RobotSystem class is used to transmit robot events, such as:

- Environment Safety Status
- Robot Touch Event
- Robot Sensor Event

Sample Introduction

Create New Project

Before creating a new robot application project, please make sure that Android Studio and AvatarMind SDK are properly set up. Create a new project and import the RobotSystem package: android.robot.hw.RobotSystem. A sample project main activity is shown below:



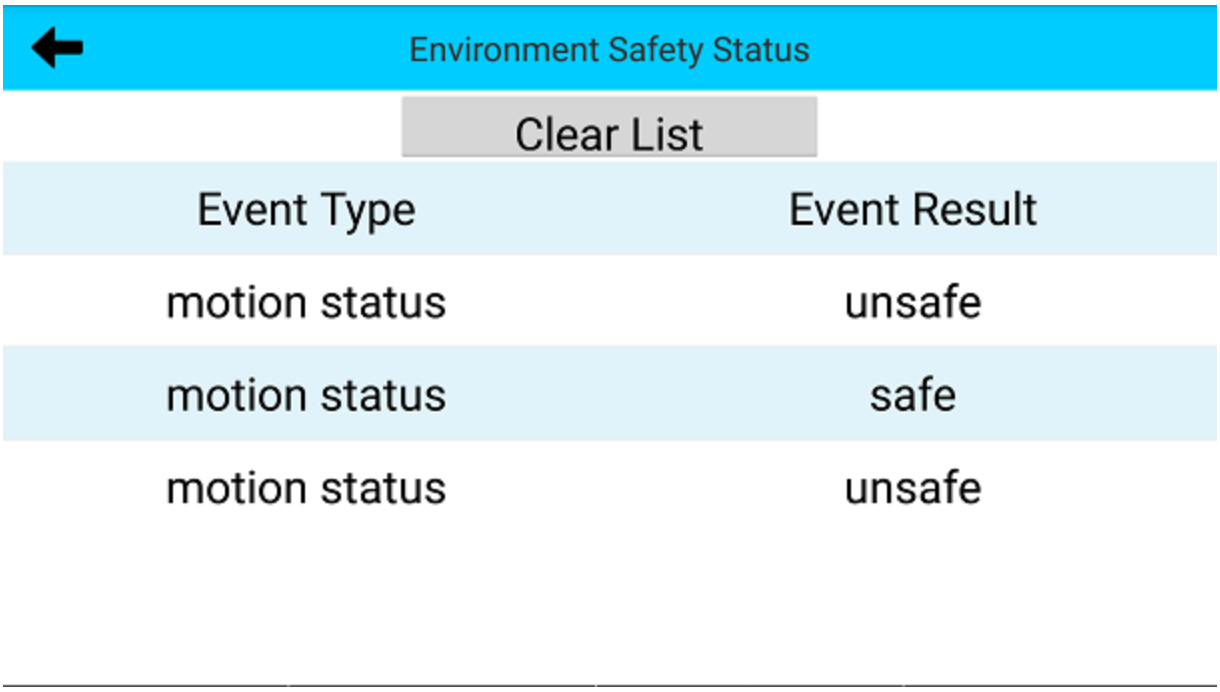
Within the main activity, three demos can be selected:

- Environment Safety Status: get the safety status of the surrounding environment
- Touch Event: get robot touch events, including head, arms, and sides
- Sensor Event: get robot sensor events, including touch, release, and collisions

Environment Safety Status

iPal can return its perceived safety for movement. If iPal should be able to move freely, it will return that the environment is safe, and if it senses obstructions or other issues, it will return that the environment is unsafe.

Sample activity for Environment Saftey Status is shown below:



Operation Process

1. Activates the listener for robot movement security when entering this activity.
2. Robot movement security events are shown in the below list:
   Event Type: type of robot movement security event
   Event Result: result of robot movement security event
3. Click the [Clear List] button to clear all data in the list.

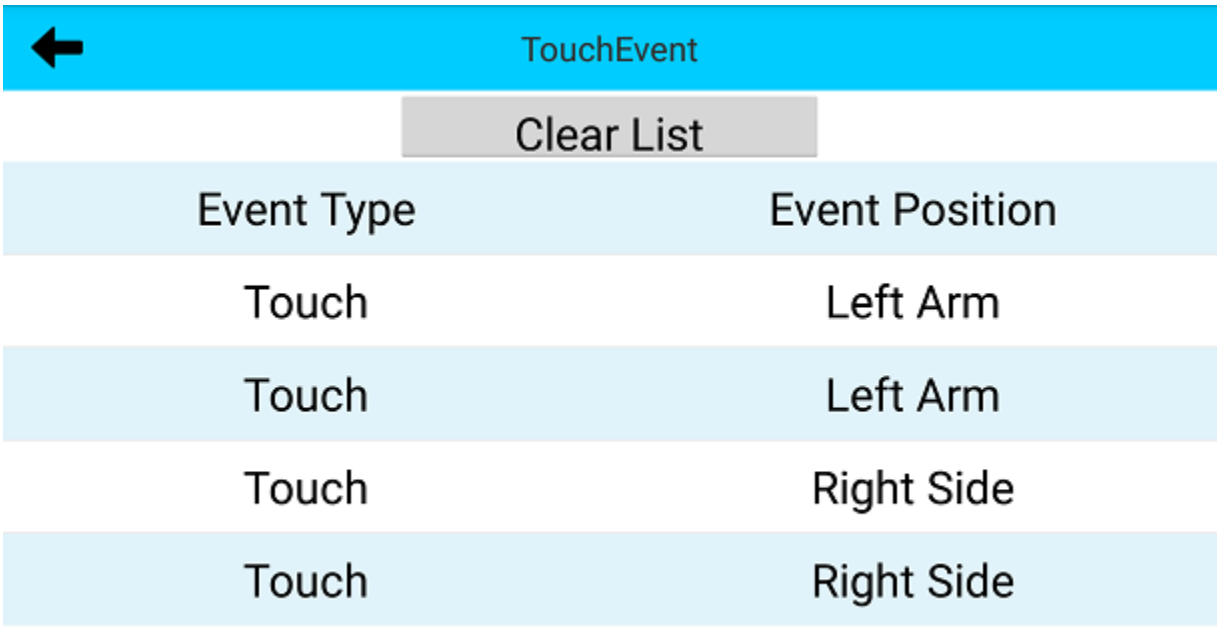
Sample Code

```
// Set a listener for robot movement security events
private RobotSystem.OnResult onResult = new RobotSystem.OnResult() {
  @Override
  public void onCompleted(int session_id, int result, int error_code) {
    if(result != 1 || error_code != 0) {
      Toast.makeText(MotionSafeActivity.this,"execute command error", Toast.LENGTH_LONG).show();
    }
  }
};

private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if (mName == 0) {
      return;
    }
    if (mName == ENVIRONMENT_SAFETY_STATUS) {
      listingMotionSafe(from, what, arg1);
    }
    // ...
  }
};

private RobotSystem mRobotSystem;
mRobotSystem = new RobotSystem();
mRobotSystem.registerListener(listener);
```

```
// Activate detection for robot movement security, if turned off, system will not transmit robot movement security event
// onResult: used to tell whether command RM_ENABLE_MOTION_SAFE_DETECT is successfully operated
mRobotSystem.setEnable(RobotDevices.DeviceType.MOTION_SAFE, 0, 1, onResult);

// ClearList:
list.clear();
adapter.notifyDataSetChanged();
```

Robot Touch Event

Sample activity for Touch Event is shown below:



Operation Process

1. Activates listener for robot touch event when entering this activity.
2. Robot touch events are shown in the below list:
   Event Type: type of robot touch event
   Event Position: result of robot touch event
3. Press the [Clear List] button to clear all data in the list.

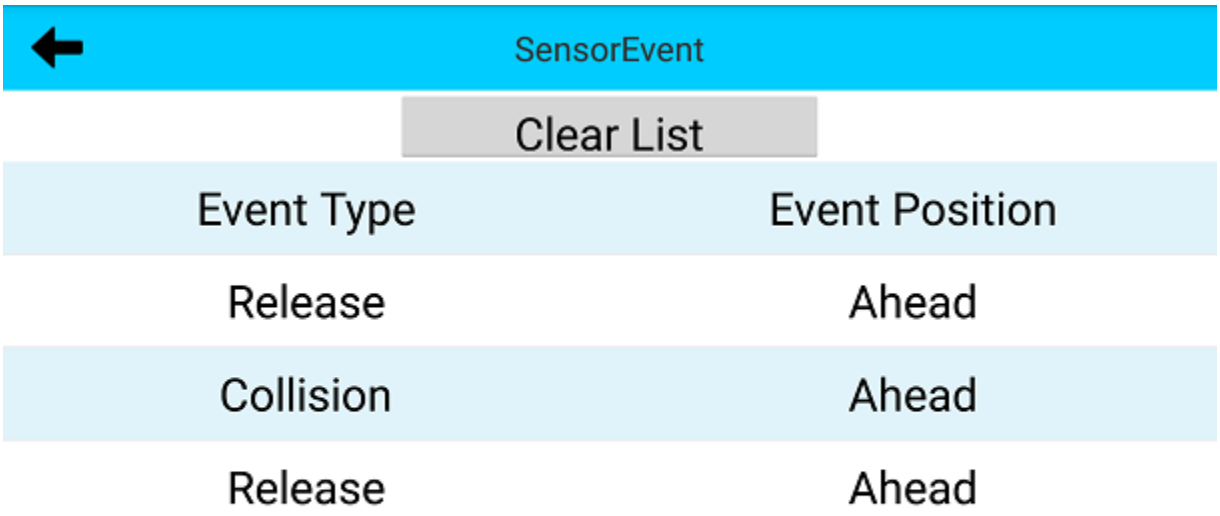
Sample Code

```
// Set a listener for robot touch events
private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if (mName == 0) {
      return;
    }
    if (mName == TOUCH_EVENT) {
      listingMotionSafe(from, what, arg1);
    }
    // ...
```

```
  }
};

private RobotSystem mRobotSystem;
mRobotSystem = new RobotSystem();
mRobotSystem.registerListener(listener);

// ClearList:
list.clear();
adapter.notifyDataSetChanged();
```

Robot Sensor Event

Sample activity for Sensor Event is as below:



Operation Process

1. Activates listener for robot sensor event when entering this page.
2. Robot sensor events are shown in the below list:
   Event Type: type of robot sensor event
   Event Position: result of robot sensor event
3. Press the [Clear List] button to clear all data in the list.

Sample Code

```
// Set a listener for robot sensor events
private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if (mName == 0) {
      return;
    }
    if (mName == SENSOR_EVENT) {
      listingMotionSafe(from, what, arg1);
```

```
    }
    // ...
  }
};

private RobotSystem mRobotSystem;
mRobotSystem = new RobotSystem();
mRobotSystem.registerListener(listener);

// ClearList:
list.clear();
adapter.notifyDataSetChanged();
```

Full Source

The full source for this tutorial is available **here**.

Overview

The RobotMotion class provides methods to control robot motors and movements. This document will introduce constant, interface, and method definitions for the RobotMotion class.

Constants

Emoji

The Emoji inner class defines constants for robot facial expressions.

Expression	Note
CLEAR	Clear expression panel
SMILE	Smile
SAD	Sad
LAUGH	Laugh
SURPRISE	Surprise
CRY	Cry
DOUBT	Doubt

Expression	Note
SHH	Do not talk
SHY	Shy
COVER_SMILE	Cover month smile
GRIMACE	Grimace
NAUGHTY	Naughty
HEARTED	Hearted
ANGRY	Angry
THINKING	Thinking
POWER_ON	Power on
POWER_OFF	Power off
WAKE_UP	Wake up
SLEEP	Sleep
TALK	Talk
LISTEN	Listen
DEFAULT	Default status
BLINK	Blink
EYECLOSE	Close eye
EYEOPEN	Open eye
FROWN	Frown

Expression	Note
EYEBINDONE	First sight
INDIFFERENT	Indifferent

Action

The Action inner class defines constants describing pre-defined robot actions.

Action	Note
SHAKE_HANDS	handshake
WAVE	Wave hands
CHEER	Cheer
RUN	Run
CLAP	Clap
AKIMBO	Akimbo
SALUTE	Salute
FOLDARM	Fold arm
FLYKISS	Throw kiss
HIGHFIVE	High five
HUG	Hug
KISS	Kiss
YES	Yes
NO	No

Action	Note
YE	Ye
THANK1	Thanks 1
THANK2	Thanks 2
THANK3	Thanks 3
LAUGH	Laugh
LISTEN	Listen
LOOK1	Look 1
LOOK2	Look 2
WORRY	Worry
SHY	Shy
TELL	Tell
ME	Me
WE	We
HANDSBACK	Hands back
WAKE	Wake up
FRIGHTENED	Frightened
TICKLE_RIGHT	Tickle right
TICKLE_LEFT	Tickle left
PUSH_FORWARD	Move forward

Action	Note
INCOMING_CALL	Call incomes
MJ_DANCE_POSE	MJ classic dance move
DANCE_POSE	Ballet dance move
TAKE_PHOTO	Take photo
UPGRADE	Upgrade
TURN_BOOK	Turn book
STOP_TALKING	Stop talking
DON_TTOUCH_ME	Do not touch me
OH_YEAH	Yeah
TO_FOLLOW	Prepare to follow
FOLLOWING	Following
WIPE_PERSPIRATION	Wipe sweat
RAIN	Raining
SNOW	Snowing
SELF_PROTECTION	Self protection
NARRATE	Narrate
IDLE	Idle status
CHAT_1	Chat 1
CHAT_2	Chat 2

Action	Note
CHAT_3	Chat 3
CHAT_4	Chat 4
PLEASE	Please

Interfaces

Listener

```
public static interface Listener {
  void onCompleted(int session_id, int result, int errorcode);
}
```

Summary
The Listener interface defines the onCompleted callback for each RobotMotion command. The session_id parameter identifies which command is currently operating. Results of this command are passed to the onCompleted method, and the implementation of onCompleted will handle the results.

Parameters

- session_id: command return value, each command in RobotMotion is assigned with a unique session_id
- result: status of the command
- errorcode: see below

errorcode

- 0: success
- -1: input parameter error
- -2: time out
- -3: command is forced to be cancelled
- -4: sending command failed

OnResult

```
public static interface OnResult {
  public void onCompleted(int id, int angle, int direction, int speed);
}
```

Summary
The OnResult interface serves as the listener for robot command operating results. If a command executes successfully (the onCompleted callback of the Listener interface returns successfully), this interface's onCompleted callback will handle data for the motor referred to by id.

Parameters

- id: motor ID, refer to RobotDevices - Motors
- angle: motor's angle
- direction: -1 negative; 0 still; 1 positive
- speed: motor's speed

Methods

finalize

```
public void finalize()
```

Summary
Destroys the RobotMotion instance and releases related memory resources.

setListener

```
public int setListener(Listener listener)
```

Summary
Registers a listener to the RobotMotion object.

Parameters

- listener: Listener object

Return

- 0 - success
- Any other number - failure

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int mSessionID = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
       if (mSessionID  == session_id) {
       // ...
       }
  }
};

mRobot.setListener(listener);
mSessionID  = mRobot.doAction(Action.WAVE);
```

getStatus

```
public int getStatus(int deviceId, OnResult onResult)
```

Summary
Gets the status of the motor referenced by deviceId.

Parameters

- deviceId: if ID is a motor(Refer to RobotDevices - Motors), this method gets the status of a single motor. If ID is a unit(Refer to RobotDevices - Units), this method gets the status of each motor in the unit.
- onResult: motor data is passed to the onCompleted method of onResult

Return

- 0 - success
- Any other number - failure

Sample Code

```
import android.robot.hw.RobotDevices.Motors;
private RobotMotion mRobot = new RobotMotion();
private RobotMotion.OnResult onResult= new RobotMotion.OnResult () {
  @Override
  public void onCompleted(int id/*motor id*/, int int angle, int direction, int speed) {
    }
};

mRobot.getStatus(Motots.ARM_ROTATION_RIGHT, onResult);
```

stop

```
public int stop()
```

Summary
Immediately stops all currently running robot actions.

Return

- 0 - success
- Any other number - failure

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
// stop all actions immediately
mRobot.stop();
```

reset

```
public int reset(int id)
```

Summary
Resets the motor(s) referenced by id to default status.

Parameters

- id: id of robot motor or set of motors(Refer to RobotDevices - Units), for example, ALL_MOTORS resets all motors to default status

Return

- 0 - success
- Any other number - failure

Sample Code

```
import android.robot.hw.RobotDevices.Units;
private RobotMotion mRobot = new RobotMotion();
// reset all motors
mRobot.reset(RobotMotion.Units.ALL_MOTORS);
```

doAction

```
public int doAction(int id)
```

Summary
Asynchronously runs the assigned internal action on robot.

Parameters

- id: **action**

Return
session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
        // action Cheer is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// Action Cheer
```

```
msession_id = mRobot.doAction(RobotMotion.Action.CHEER);
```

doAction

```
public int doAction(int id, int flags)
```

Summary
Runs the assigned internal action on robot.

Parameters

- id: **action**
- flags: refer to **flags**

Return
session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id_1 = 0;
private int msession_id_2 = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id_1 == session_id) {
          // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// do cheer action, then do wave action
msession_id_1 = mRobot.doAction(RobotMotion.Action.CHEER, 0x80);
msession_id_2 = mRobot.doAction(RobotMotion.Action.WAVE, 0x80);
```

startMotor

```
public int startMotor(byte[] pdata, int len)
```

Summary
Asynchronously controls movement of a motor.

Parameters

- pdata: please refer to the **pdata syntax**
- len: length of pdata

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private byte[] bytes = new byte[10];
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// Rotate right arm 100 degrees
bytes[0] = 0xF1;
bytes[1] = 0x8E;
bytes[3] = RobotMotion.Motors.ARM_ROTATION_RIGHT;
bytes[4] = (byte)(100 & 0xFF);
bytes[5] = (byte)((100 >> 8) & 0xFF);
bytes[6] = (byte)0x00;
bytes[7] = (byte)(1000);
bytes[8] = (byte)0x00;
bytes[9] = (byte)0x00;
mRobot.startMotor(bytes, sizeof(bytes));
```

startMotor

```
public int startMotor(byte[] pdata, int len, int flags)
```

Summary
Asynchronously controls movement of a motor. Duration of the action is set in pdata.

Parameters

- pdata: please refer to the **pdata syntax**
- len: length of pdata
- flags: refer to **flags**

Return

- session_id of this command

startMotor

```
public int startMotor(byte[] pdata, int len, int duration, int flags)
```

Summary
Controls movement of motor.

Parameters

- pdata: please refer to the **pdata syntax**
- len: length of pdata
- duration: duration of movement (unit is 1ms)
- flags: refer to **flags**

Return

- session_id of this command

startMotor

```
public int startMotor(int id, int angle, int duration, int flags)
```

Summary
Controls movement of motor.

Parameters

- id: assigned robot motor(Refer to RobotDevices - Motors)
- angle: motor's absolute angle
- duration: duration of movement (unit is 100ms)
- flags: refer to **flags**

Return

- session_id of this command

Sample Code

```
import android.robot.hw.RobotDevices.Motors;
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// Robot right arm rotate to 50 degrees
msession_id = startMotor(RobotDevices.Motors.ARM_ROTATION_RIGHT, 100, 1000, 0x00);
msession_id = startMotor(RobotDevices.Motors.ARM_ROTATION_RIGHT, 50, 1000, 0x00);
Sleep(10);
// Robot right arm rotate 100 degrees and then back to 50 degrees
msession_id = startMotor(RobotDevices.Motors.ARM_ROTATION_RIGHT, 100, 1000, 0x80);
msession_id = startMotor(RobotDevices.Motors.ARM_ROTATION_RIGHT, 50, 1000, 0x80);
```

nodHead

```
public int nodHead()
```

Summary
Nods the robot's head.

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
msession_id = mRobot.nodHead();
```

shakeHead

```
public int shakeHead()
```

Summary
Shakes the robot's head.

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
msession_id = mRobot.shakeHead();
```

startWalk

```
public int startWalk()
```

Summary
Makes the robot walk until it reaches a barrier.

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion ();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener () {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
        if(msession_id == session_id){
          // action is completed
          // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
        }
  }
};
mRobot.setListener (listener);
msession_id = mRobot.startWalk();
```

startWalk

```
public int startWalk(int distance, int speed, int flags)
```

Summary
Makes the robot walk the designated distance, either forward or backwards.

Parameters

- distance: robot walking distance (unit is mm)
- speed: 1 slow; 2 medium; 3 fast
- flags: refer to **flags**

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id){
```

```
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// slowly walk 1m forward
msession_id = mRobot.startWalk(1000, 1, 0x00);
```

stopWalk

```
public int stopWalk(int flags)
```

Summary
Makes the robot immediately stop walking.

Parameters

- flags: refer to **flags**

Return

- 0 - success
- Any other number - failure

Sample Code

```
private RobotMotion mRobot = new RobotMotion ();
// Stop walking immediately
mRobot.stopWalk(0x01);
```

turn

```
public int turn(int angle, int speed)
```

Summary
Asynchronously controls turning of wheels.

Parameters

- angle: turning angle
- speed: 1 slow; 2 medium; 3 fast

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// turn 100 degree in medium speed
msession_id = mRobot.turn(100, 2);
```

turn

```
public int turn(int angle, int speed, int flags)
```

Summary
Controls turning of wheels.

Parameters

- angle: turning angle
- speed: 1 slow; 2 medium; 3 fast
- flags: refer to **flags**

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id){
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener (listener);
// Turn robot 100 degrees at medium speed
msession_id = mRobot.startWalk(1000, 1, 0x00);
msession_id = mRobot.turn(100, 2, 0x00);
Sleep(10);
// Robot walks 1m then turns 100 degrees at medium speed
msession_id = mRobot.startWalk(1000, 1, 0x80);
msession_id = mRobot.turn(100, 2, 0x80);
```

emoji.

```
public int emoji(int id)
```

Summary
Asynchronously controls robot facial expression.

Parameters

- id: **expression**

Return

- session_id of this command

Sample Code

```
private RobotMotion mRobot = new RobotMotion();
private int msession_id = 0;
private RobotMotion.Listener listener = new RobotMotion.Listener() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if(msession_id == session_id) {
      // action is completed
      // Note: mRobot or update UI thread cannot be accessed inside this callback, using a
Handler is advised instead
    }
  }
};

mRobot.setListener(listener);
// display cry expression
msession_id = mRobot.emoji(RobotMotion.Emoji.CRY)
```

Appendix

Definition For pdata

A pdata byte array consists of 10 bytes, including single motor info:

Byte	Note
0	Constant, 0xF1
1	Constant, 0xBE
2	Reserved
3	Motor id

Byte	Note
4	angle/distance lower 4 bits. Eg, angle & 0xFF
5	angle/distance higher 4 bits. Eg, (ang >> 8) & 0xFF
6	Motion flag, 0, slow stop; 1, quick stop; 2 keep moving when reaching position until motor limit
7	Time for motion, in unit of 100ms
8	Reserved
9	Reserved

Definition For flags

flags consist of 1 byte, including control for motion:

Bit	Note
0-1	0: slow stop; 1: quick stop; 2: keep moving when certain angle reached until reaching limit; 3 reserved
2	Reserved, 0
3	Reserved, 0
4	Reserved, 0
5	Reserved, 0
6	1: motion operated immediately
7	0: asynchronous; 1: synchronous; Note: this bit has effect only on an identical motor

Overview

This document introduces constants, interfaces, and methods for the RobotPlayer class, which represents a robot motion player. AvatarMind defines two types of media resource files, ARM (Avatar Robot Motion) and ARC (Avatar Robot Content). RobotPlayer is used to play ARM files, which contain a sequence of robot motor movements.

Constants

The ListenerResult inner class defines constants indicating RobotPlayer's status:

Constant	Note
UNREADY	Player is not ready to play
OVER	Player has finished playing motion sequence
READY	Player is ready to play a motion file
PAUSE	Player is currently paused
RUNNING	Player is currently playing

Interfaces

Listener

```
public static interface Listener {
  void onCompleted(int result, int errorcode);
}
```

Summary
The Listener interface handles status changes in RobotPlayer. When RobotPlayer changes states, the new state is passed to the onCompleted callback.

Parameters

- result: mentioned in **Constants** above
- errorcode: see below

errorcode

- 0: success
- -2: illegal motion sequence
- -3: file not found or failed to open file

Methods

setListener

```
public int setListener(Listener listener)
```

Summary
Registers a listener to the RobotPlayer object.

Parameters

- listener: Listener object

Return

- 0 - success
- Any other number - failure

setDataSource

```
public int setDataSource(String motionFile)
```

Summary
Reads in the ARM file pathname, and sets the starting frame at the beginning of the file.

Parameters

- motionFile: absolute path for ARM file

Return

- 0 - success
- Any other number - failure

setDataSource

```
public int setDataSource(String motionFile, int offset, int size)
```

Summary
Reads in the ARM file pathname, and sets the starting frame at the indicated offset.

Parameters

- motionFile: absolute path for ARM file
- offset: offset of ARM data in assigned file (in milliseconds)
- size: ARM file size

Return

- 0 - success
- Any other number - failure

setDataSource

```
public int setDataSource(byte[] bytes, int offset, int size)
```

Summary
Reads in the ARM-formatted byte stream as RobotPlayer source, and sets the starting frame at the indicated offset.

Parameters

- bytes: binary data instream
- offset: offset of ARM data in assigned file (in milliseconds)
- size: ARM file size

Return

- 0 - success
- Any other number - failure

prepare

```
public int prepare()
```

Summary
This method should be called after setting the RobotPlayer source in order to complete the player configuration.

Return

- 0 - success
- Any other number - failure

start

```
public int start()
```

Summary
This method should be called after prepare() is called. The robot will start playing the motion file.

Return

- 0 - success
- Any other number - failure

pause

```
public int pause()
```

Summary
Pauses current playing. Note that the robot's movements are paused, not reset.

Return

- 0 - success
- Any other number - failure

resume

```
public int resume()
```

Summary
If the current ARM file is paused, resumes current playing.

Return

- 0 - success
- Any other number - failure

stop

```
public int stop()
```

Summary
Stops current playing.

Return

- 0 - success
- Any other number - failure

getDuration

```
public int getDuration()
```

Summary
Gets the total duration of the current motion file from start to the end.

Return

- The duration of the ARM file (in ms)

getPosition

```
public int getPosition()
```

Summary
Gets the robot’s current progress in the ARM file.

Return
The position of the robot’s progress in the ARM file. The unit is in milliseconds.

setPosition

```
public int setPosition(int pos)
```

Summary
Sets the robot's position in the loaded ARM file. If successful, the robot will start from the assigned frame.

Parameters

- pos: position in the ARM file (in ms)

Return

- Initial frame position before setPosition changes it

setEnable

```
public int setEnable(int enable)
```

Summary
Enables or disables RobotPlayer.

Parameters

- enable:
  0: player is enabled 1: player is disabled

Return

- 0 - success
- Any other number - failure

isEnable

```
public int isEnable()
```

Summary
Retrieves whether RobotPlayer is enabled or disabled.

Return

- 0 - player is enabled
- 1 - player is disabled

Sample Code

Sample For Loading ARM Instream

```
public byte[] getFrom7.1/assets/(String fileName) {
  try {
    InputStream in = getResources().get7.1/assets/().open(fileName);
    int fileSize = in.available();
    byte[] buffer = new byte[fileSize];
    in.read(buffer);
    return buffer;
  } catch (Exception e) {
    e.printStackTrace();
  }
  return new byte[0];
}
```

Sample For Loading ARM File

```
RobotPlayer robotPlayer = new RobotPlayer();
byte[] bytes = getFrom7.1/assets/("test.arm",)
robotPlayer.setDataSource(bytes, 0, bytes.lenght);
robotPlayer.prepare();
```

```
robotPlayer. start();
```

Overview

SpeechManager has one main module:

- `TTS`(Text To Speech): converts text into spoken words.

Tips:

- Make sure to call the `setTtsEnable` method in the proper stage of the Activity life cycle.
- We suggest using a system service to initialize SpeechManager, and manually initialize SpeechManager only if the system service fails.

Sample Code

```
  private SpeechManager mSpeechManager;
  mSpeechManager = (SpeechManager) getSystemService(SpeechService.SERVICE_NAME);
  if (mSpeechManager == null) {
      mSpeechManager = new SpeechManager(this, new OnConnectListener() {
          @Override
          public void onConnect(boolean status) {
              if (status) {
                  LogUtils.d("speechManager init success!");
                  if (mSpeechManager.getTtsEnable()) {

                  }
              } else {
```

```
            }
        }
    }, "com.avatar.dialog");
}
```

General

OnConnectListener

```
public interface OnConnectListener {
  public void onConnect(boolean status);
}
```

Summary
Returns the current voice service status (enabled or disabled). Generally, voice service will be enabled when `SpeechManager` is constructed.
Callback

- onConnect

```
// status: voice service is enabled or disabled
public void onConnect(boolean status)
```

SpeechManager

```
public SpeechManager(Context context, OnConnectListener listener)
```

Summary
The SpeechManager constructor. You must create a SpeechManager object before using any related functionality of the voice module. Corresponding listeners for the modules you wish to use must also be registered to the SpeechManager object.

Parameters

- context: context in which object is created
- listener: OnConnectListener, monitors whether voice service is connected

Sample Code

```
private SpeechManager mSpeechManager;
mSpeechManager = (SpeechManager) getSystemService(SpeechService.SERVICE_NAME);
if (mSpeechManager == null) {
  mSpeechManager = new SpeechManager(this, new OnConnectListener() {
    @Override
    public void onConnect(boolean status) {
      if (status) {
        LogUtils.d("speechManager init success!");
        if (mSpeechManager.getTtsEnable()) {

        }
      } else {
```

```
      }
    }
  }, "com.avatar.dialog");
}
```

Overloaded

```
public SpeechManager(Context context) {
  this (context, (OnConnectListener) null);
}
```

isEstablished

```
public boolean isEstablished()
```

Summary
Returns whether the current voice service is enabled. When user calls any method in SpeechManager, voice service must be enabled. Generally, when SpeechManager is initialized from the system service the voice service is automatically enabled. We suggest checking the voice service status before calling any methods in SpeechManager.

Return

- boolean: status of voice service

shutdown

```
public void shutdown()
```

Summary
Disconnects current system from voice service.

setChatEnable

```
public boolean setChatEnable(final boolean enable)
```

Summary
Sets robot voice chat status. This method has high priority. If enable is false, ASR and NLU will be disabled, but TTS will remain enabled.

Parameters

- enable: enable or disable robot voice

Return

- boolean: initial robot voice chat status before setting is changed

getChatEnable

```
public boolean getChatEnable()
```

Summary
Gets robot chatting status.

Return

- boolean: robot voice chat status

TTS

TtsListener

```
public interface TtsListener {
  public void onBegin(int requestId);

  public void onError(int requestId);

  public void onEnd(int requestId);
}
```

Summary

- The TtsListener interface serves as the listener for getting TTS status. Each startSpeaking command will have a unique requestId used to distinguish between multiple speaking sentences; this ID will be passed to the TtsListener callbacks.

Callback

- onBegin

```
// Called when robot begins to process text
// requestId: text ID
public void onBegin(int requestId)
```

- onError

```
// Called when robot encounters an error when processing text
// requestId: ID of text returning an error
public void onError(int requestId)
```

- onEnd

```
// Called after robot finishes speaking
```

```
// requestId: text ID
public void onEnd(int requestId)
```

setTtsListener

```
public void setTtsListener(TtsListener listener)
```

Summary
Registers a TtsListener to a SpeechManager object. User can get TTS callbacks through this listener.

Parameters

- listener: TtsListener object

Sample Code

```
private SpeechManager mSpeechManager;
private TtsListener mTtsListener = new TtsListener() {
  @Override
  public void onBegin(int requestId) {

  }

  @Override
  public void onEnd(int requestId) {

  }

  @Override
  public void onError(int requestId) {

  }
};

//......

mSpeechManager.setTtsListener(mTtsListener);
```

setTtsEnable

```
public boolean setTtsEnable(final boolean enable)
```

Summary
Set TTS status to enabled or disabled.

Parameters

- enable: status (enabled or disabled)

Return

- boolean: inital TTS status before current setting

getTtsEnable

```
public boolean getTtsEnable()
```

Summary
Gets current TTS status.

Return

- boolean: TTS status (enabled or disabled)

isSpeaking

```
public boolean isSpeaking()
```

Summary
Returns whether robot is speaking through TTS.

Return

- boolean: whether robot is currently speaking

Override

```
// requestId: whether robot is speaking content with this ID
public boolean isSpeaking(final int requestId)
```

startSpeaking

```
public int startSpeaking(final String text, final boolean isRealtime, final boolean
isShowSubtitle)
```

Summary
Makes robot begin speaking through TTS.

Parameters

- text: content to be spoken
- isRealtime: if true, will stop any current voice commands and begin speaking new input text immediately
- isShowSubtitle: if true, will show subtitles while speaking

Return

- int: requestId of this command, this id will be used in `TTSListener`

Override

```
// equivalent to startSpeaking(text, false, false)
public int startSpeaking(String text)

// equivalent to startSpeaking(text, isRealtime, false)
public int startSpeaking(String text, boolean isRealtime)
```

forceStartSpeaking

```
public int forceStartSpeaking(final String text, final boolean isRealtime, final boolean
isShowSubtitle)
```

Summary
When robot is speaking, even if close the TTS through setTtsEnable, robot can still speak through TTS by this method.

Parameters

- text: content to be spoken
- isRealtime: if true, will stop any current voice commands and begin speaking new input text immediately
- isShowSubtitle: if true, will show subtitles while speaking

Return

- int: requestId of this command, this id will be used in `TtsListner`

Override

```
// equivalent to forceStartSpeaking(text, false, false)
public int forceStartSpeaking(String text)
```

stopSpeaking

```
public boolean stopSpeaking(final int requestId)
```

Summary
Makes robot stop speaking through TTS.

Parameters

- requestId: stop speaking content with corresponding id. If id is -1, stops all content in the TTS queue.

Return

- boolean: status of this command (success or failure)

getTtsSpeed

1.

```
public int getTtsSpeed()
```

Summary
Get TTS speed.

Return
Return the value of TTS speed.

setTtsSpeed

1.

```
public int setTtsSpeed(int speed)
```

Summary
Set the current TTS speed.

Parameters

- speed: the value of TTS speed, ranging from 0 to 100.

Return
Return -1 if an error occurs or the system TTS does not support speed adjustment, otherwise the speed value before setting is returned.

2.

```
public int setTtsSpeed(final int speed, final boolean persist)
```

Summary
Set the specified TTS speed.

Parameters

- speed: the value of TTS speed, ranging from 0 to 100.
- persist: whether to save the configuration, return true to use the current TTS speed after restart, false to restore the system default TTS speed after restart.

Return
Return -1 if an error occurs or the system TTS does not support speed adjustment, otherwise the speed value before setting is returned.

Overview

This document will introduce constant definitions, API and system event processing in RobotSystem class.

Constants

The CallbackCommand inner class defines the constants shown below:

Robot Event Types

Constant	Note
RC_EVENT_TYPE	Motion control event, including motion security event
RP_EVENT_TYPE	Battery event
RF_EVENT_TYPE	Expression event
RB_EVENT_TYPE	Motor event
RM_EVENT_TYPE	System event
RC_SENSOR_TYPE	Sensor event

Touch Event

Constant	Note
RF_EVENT_TOUCH	Touch event
RF_EVENT_SOURCE	Reserved
RF_EVENT_LONG_TOUCH	Robot is touched for at least 1s
RF_EVENT_RELEASE	Touch released

Definitions For Touch Area

Constant	Note
RF_HEAD_TOUCH	Head
RF_LEFT_SHOULDER_TOUCH	Left arm

Constant	Note
RF_RIGHT_SHOULDER_TOUCH	Right arm
RF_LEFT_OXTER_TOUCH	Left side
RF_RIGHT_OXTER_TOUCH	Right side

Sensor Event

Constant	Note
RC_FRONT_UPPER_OBSTACLE	Barrier at upper front
RC_FRONT_LOWER_OBSTACLE	Barrier at lower front
RC_BACK_UPPER_OBSTACLE	Barrier at upper back
RC_BACK_LOWER_OBSTACLE	Barrier at lower back
RC_LEFT_OBSTACLE	Barriers on the left
RC_RIGHT_OBSTACLE	Barriers on the right
RC_FORWARD_FALL	Fall forward
RC_BACKWARD_FALL	Fall backward
RC_FRONT_COLLISION	Collision at front
RC_ENVIRONMENT_OBSTACLE	Barrier at surroundings
RC_APPROACH_SLOW	Approach slowly
RC_APPROACH_FAST	Approach quickly
RC_GO_AWAY	Keep away from
RC_TOO_CLOSE	Barrier approaching

Constant	Note
RC_FRONT_COLLISION_RELEASE	Release collision

Definition For Sensor Area

Constant	Note
EVENT_FRONT	Forward
EVENT_BACK	Backward
EVENT_LEFT	Left
EVENT_RIGHT	Right

Motion Security Event

Constant	Note
RC_CHANGE_MOTION_SAFE_STATUS	Motion security status

Result For Motion Security Event

Constant	Note
ROBOT_MOTION_SAFE	Motion safe
ROBOT_MOTION_UNSAFE	Motion not safe

Interfaces

Listener

```
public static interface Listener {
  void onMessage(int from, int what, int arg1, int arg2);
}
```

Summary
The Listener interface transmits all events to the onMessage callback. **Parameters**

- from: type of robot event, please refer to **Robot Event Types**.
- what: sub-type of specific robot event, please refer to **touch event**, **sensor event** and **motion security event**.
- arg1: sub-type parameters of robot event, please refer to **touch area**, **sensor area** and **results of motion security**.
- arg2: reserved

OnResult

```
public static interface OnResult {
  void onCompleted(int session_id, int result, int errorcode);
}
```

Summary
Listener for robot motions or actions. According to session_id, user can identify the current robot command and whether it has completed or failed from this callback.

Parameters

- session_id: ID for each command
- result: result of current command
- errorcode: 0 is success, others are failure

Methods

finalize

```
public void finalize()
```

Summary
Destroys the RobotSystem instance and releases related memory resources.

registerListener

```
public int registerListener(Listener listener)
```

Summary
Registers a Listener to RobotSystem object. Callbacks for transmitting event will be handled by this listener.

Parameters

- listener: Listener object

Return

- 0 - success
- Any other number - failure

setEnable

```
public int setEnable(int deviceType, int deviceId, int enable, OnResult result)
```

Summary
Sets status for the target device.

Parameters

- deviceType: type of robot device (Refer to RobotDevices - DeviceType)
- deviceId: if the ID is for a motor, sensor or expression(Refer to RobotDevices - Motors, Sensors and EMOJIS), this method enables or disables a single motor or sensor. If ID is a unit(Refer to RobotDevices - Units), all devices in the unit are enabled or disabled.
- enable: 1 - on, 0 - off
- result: callback to OnResult object for result

Return

- session_id of this command

Sample

```
import android.robot.hw.RobotDevices.DeviceType;
import android.robot.hw.RobotDevices.Motors;

private RobotSystem mRobotSystem = new RobotSystem();
private int mSessionID = 0;
private RobotSystem.OnResult onResult= new RobotSystem.OnResult() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if (mSessionID  == session_id) {
      if(errorcode == 0) {

      }
    }
  }
};

mSessionID = mRobotSystem.setEnable(RobotDevices.DeviceType.MOTOR,
RobotDevices.Motors.ARM_ROTATION_RIGHT, 1, onResult);
// Note: if deviceType is DeviceType.TOUCH_LISTENER, return value of setEnable(...) is always 0
```

isEnable

```
public int isEnable(int deviceType, int deviceId, OnResult result)
```

Summary
Returns the status of target device; the device status will be passed to the OnResult callback.

Parameters

- deviceType: type of robot device(Refer to RobotDevices - DeviceType)

- deviceId: if the ID is for a motor, sensor or expression(Refer to RobotDevices - Motors, Sensors and EMOJIS), this method checks a single motor or sensor. If ID is a unit(Refer to RobotDevices - Units), this method checks the entire unit.
- result: OnResult object which status is passed to

Return

- session_id of this command

Sample

```
import android.robot.hw.RobotDevices.DeviceType;
import android.robot.hw.RobotDevices.Motors;

private RobotSystem mRobotSystem = new RobotSystem();
private int mSessionID = 0;
private RobotSystem.OnResult onResult = new RobotSystem.OnResult() {
  @Override
  public void onCompleted (int session_id, int result, int errorcode) {
    if (mSessionID  == session_id) {
      if(errorcode == 0) {
        // result : 0 - off, 1 - on
      }
    }
  }
};

mSessionID = mRobotSystem.isEnable(RobotDevices.DeviceType.MOTOR,
RobotDevices.Motors.ARM_ROTATION_RIGHT, onResult);
// Note: if deviceType is DeviceType.TOUCH_LISTENER, return value of isEnable(...) is always 1
```

getRobotSex

```
public String getRobotSex()
```

Summary
Get robot's gender.

Return

- true - male
- false - female

Sample Code

Listener For Touch Event

```
private RobotSystem robotSystem;
private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if(from == RobotSystem.CallbackCommand.RF_EVENT_TYPE) {
      // what: type of the sensor event
```

```
      // arg1: refers to touch area
      // arg2: reserved
    }
  }
};

robotSystem = new RobotSystem();
robotSystem.setListener(listener);
```

Sensor Event Sample

```
private RobotSystem robotSystem;
private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if(from == RobotSystem.CallbackCommand.RC_SENSOR_TYPE) {
      // what: type of the sensor event
      // arg1: refers to sensor area
      // arg2: reserved
    }
  }
};

robotSystem = new RobotSystem();
robotSystem.setListener(listener);
```

Security Event Sample

```
private RobotSystem robotSystem;
private RobotSystem.Listener listener = new RobotSystem.Listener() {
  @Override
  public void onMessage(int from, int what, int arg1, int arg2) {
    if(from == RobotSystem.CallbackCommand.RC_EVENT_TYPE) {
      // what: type of the security event
      // arg1: result of the security event
      // arg2: reserved
    }
  }
};

private RobotSystem.OnResult onResult = new RobotSystem.OnResult() {
    @Override
    public void onCompleted(int session_id, int result, int error_code) {

    }
};

robotSystem = new RobotSystem();
robotSystem.setListener(listener);
```

```
robotSystem.setEnable(DeviceType.MOTION_SAFE, 1, onResult);
```

Overview

The RobotDevices class defines constants for controlling your robot device. This document will introduce these definitions in the RobotDevices class.

Constants

DeviceType

The DeviceType inner class defines the types of components in the robot.

Constant	Note
MOTOR	Motors to trigger each joint
SENSOR	Sensors on robot
TOUCH_LISTENER	Touch device
MOTION_SAFE	Chassis
EMOJI	Expression panel

Units

The Units inner class defines sets of robot body parts and sensors.

Constant	Note
ALL_MOTORS	All motors
ARMS_MOTORS	Both arms
RIGHT_ARM_MOTORS	Right arm
LEFT_ARM_MOTORS	Left arm
WHEEL_MOTORS	Wheel

Constant	Note
NECT_MOTORS	Neck
ALL_SENSORS	All sensors
EMOJI_MOTORS	Emoji
WAIST_MOTORS	Waist

Sensors

The Sensor inner class defines the robot's body sensors.

Constant
IRDA_BOTTOM_FRONT_LEFT
IRDA_BOTTOM_FRONT_RIGHT
IRDA_BOTTOM_BACK_CENTER

Constant	Note	Angle
ARM_ROTATION_RIGHT	Rotate right arm	[-25, 175]

Constant		
ARM_SWING_RIGHT	Swing right arm	[0, 65]
FOREARM_ROTATION_RIGHT	Rotate right forearm	[-80, 80]
FOREARM_SWING_RIGHT	Swing right forearm	[0, 90]
WRIST_RIGHT	Rotate right wrist	[-80, 80]
ARM_ROTATION_LEFT	Rotate left arm	[-25, 175]
ARM_SWING_LEFT	Swing left arm	[0, 65]
FOREARM_ROTATION_LEFT	Rotate left forearm	[-80, 80]
FOREARM_SWING_LEFT	Swing left forearm	[0, 90]
WRIST_LEFT	Rotate left wrist	[-80, 80]
WHEEL_LEFT	Wheel turn left	
WHEEL_RIGHT	Wheel turn right	
NECK_ROTATION	Rotate neck	[-50, 50]
NECK_TILT	Tilt neck	[-10. 28]
ULTRASONIC_FRONT_BELLY		
ULTRASONIC_BACK_WAIST		
ULTRASONIC_LEFT_ANKLE_OUTSIDE		

Constant
ULTRASONIC_RIGHT_ANKLE_OUTSIDE
ULTRASONIC_CHASSIS

Motors

The Motors inner class defines the robot's motors and the min/max angles for each motor.

EMOJIS

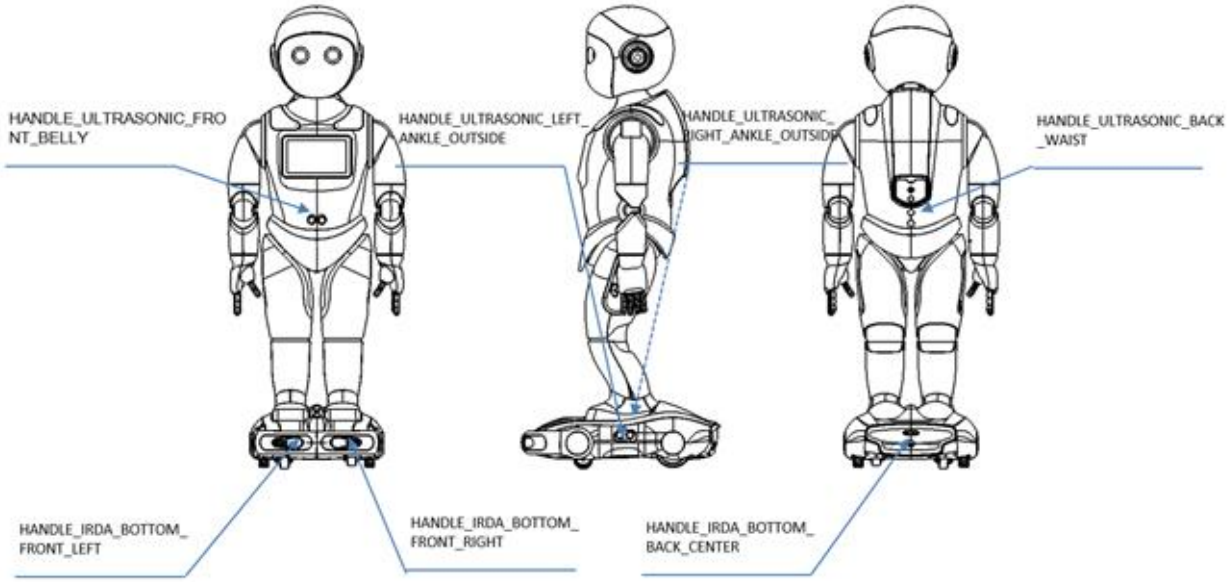
The EMOJIS inner class defines sections on the robot's expression panel.

Expression	Note
FACE	All expression on robot face panel
EYE	Robot's eye

Overview

The Sensor class defines constants for robot sensors and provides an extended API for relevant sensors.

Robot Sensor Distribution



Constants

Constant	Note
TYPE_IRDA_DISTANCE	acquire all infrared sensors on robot
TYPE_ULTRASONIC_DISTANCE	acquire all ultrasonic sensors on robot
HANDLE_IRDA_BOTTOM_FRONT_LEFT	infrared sensor at left of robot chassis
HANDLE_IRDA_BOTTOM_FRONT_RIGHT	infrared sensor at right of robot chassis
HANDLE_IRDA_BOTTOM_BACK_CENTER	infrared sensor at middle back of robot chassis
HANDLE_ULTRASONIC_FRONT_BELLY	ultrasonic sensor at robot front belly
HANDLE_ULTRASONIC_BACK_WAIST	ultrasonic sensor at robot back waist
HANDLE_ULTRASONIC_LEFT_ANKLE_OUTSIDE	ultrasonic sensor at outward of robot left ankle

Constant	Note
HANDLE_ULTRASONIC_RIGHT_ANKLE_OUTSIDE	ultrasonic sensor at outward of robot right ankle

```

## Trigger Amazon Alexa login page from 3rd-party applications

3rd-party application can use the sample code below to trigger starting Amazon Alexa login page before using Alexa voice engine. Page will be redirected to Amazon Alexa login page from external applications.

## Sample code

```
  Intent intent = new Intent("avatar.intent.action.LOGIN_WITH_AMAZON");
  intent.addCategory(Intent.CATEGORY_DEFAULT);
  try {
          startActivity(intent);
  } catch (ActivityNotFoundException e) {
          Log.i(TAG, "No activity to handle alexa login");
  }
```

## Trigger Amazon Alexa voice interaction feature from 3rd-pary applications

Generally user has to use wake word strategy in opening iPal conversational feature of Amazon Alexa, specifically by speaking certain wake word to trigger iPal under the listening mode. However, for external 3rd-paty applications, developers can send broadcast to enable this feature.

## Sample code

```
  /**
   * Broadcast action: simulate wake word has been detected.
   */
  private static final String ACTION_WAKE_WORD_DETECTED =
"avatar.intent.action.WAKE_WORD_DETECTED";

  /**
   * Intent extra: specify a wake word.
   */
  private static final String EXTRA_KEY_WAKE_WORD = "wake_word";

  // No assigning wake word
  Intent intent = new Intent("avatar.intent.action.WAKE_WORD_DETECTED");
  intent.addCategory(Intent.CATEGORY_DEFAULT);
  sendBroadcast(intent);
```
```

```
// Assigning wake word
Intent intent = new Intent("avatar.intent.action.WAKE_WORD_DETECTED");
intent.addCategory(Intent.CATEGORY_DEFAULT);
intent.putExtra("wake_word", "Alexa");
sendBroadcast(intent);
```

Note

Developers can assign the wake word in their own applications via input parameter EXTRA_KEY_WAKE_WORD. This value will ONLY be effective under the condition of setting wake word to “Auto” in iPal <*Settings-Robot Settings-Voice*>.
Currently the supported values for EXTRA_KEY_WAKE_WORD are “Alexa” and “Hey iPal”. If EXTRA_KEY_WAKE_WORD value is not assigned or assigned value is neither “Alexa” nor “Hey iPal”, system will use the default value in iPal <*Settings-Robot Settings-Voice*>.

Sample

If wake word in iPal <*Settings-Robot Settings-Voice*> is “Alexa”, while value of EXTRA_KEY_WAKE_WORD is “Hey iPal”, system will use “Alexa” by default.
Only if wake word in iPal <*Settings-Robot Settings-Voice*> is “Auto”, and value of EXTRA_KEY_WAKE_WORD is “Hey iPal”, system will use “Hey iPal” as valid value.

Introduction

Robot vision service module includes:

- Face registration
- Face recognition
- Face management

Face registration

Face registration: Photo registration, video registration, photo import.

- Photo registration and video registration support automatic registration and manual registration.
- Automatic registration supports a naming convention using ″_" to split information.
- Photo registration supports multiple photos for a single registration; the character immediately after the last ″_" must be a numeric digit (0-9), indicating individual images.
- The face registration nicknames are not unique, but full names are unique.
- Face registration feedback is returned to the original application in real time.

Face recognition

Face recognition: information security.

- Only the application that registered a face can receive that face’s full name. Other applications only receive the face’s nickname.
- Face recognition results include confidence level.

- Supports external applications opening the camera: open the camera then call face recognition; send the video stream to the vision service through its interface; the vision service processes the video stream and returns recognition information to the external application.
- Supports vision service opening the camera: only open face recognition; vision service opens the camera; vision service processes the video stream and returns recognition information to external applications.
- External applications are strictly forbidden from opening face recognition in the background or in a service; it will lead to a camera conflict. Please open face recognition in the onResume lifecycle of your Activity.

Face management

Face management: permissions security.

- Only the external application that registered the face has management authorization; other applications do not have authorization.

Usage

1.Please call the RobotVisionClient method in the life cycle of Activity.
2.Please use RobotVisionClient in **the commercial version of the robot (version 8.0.8 and above)**.
3.Please refer to the function demo code **RobotVisionClient2.zip**, please **download** it at the corresponding address.
4.For further questions, please consult:
haizhou.wu@avatarmind.com, 420660135@qq.com, 15996211983.

Initialization

FaceEventListener

```
public interface FaceEventListener {
    void onConnectionStatus(boolean isConnected);
    void onVisionEvent(String event);
    void OnRegisterEvent(String event);
}
```

Summary

1.Used to determine whether vision services are connected successfully.
2.For receiving face recognition events.
3.For receiving face registration events.

Callback

- onConnectionStatus

```
// isConnected: whether vision services are connected
```

```
public void onConnectionStatus(boolean isConnected)
```

- onVisionEvent

```
// event: face recognition events
public void onVisionEvent(String event)
// rvf;vision;event;
// face;                  : Type          (VisionEvent.getType())
// yes,                   : yes or no     (yes: event continue; no: event end)
// friend_1,              : Nickname      (FaceEvent.getNickname)
// xxx.friend_1,          : Fullname      (FaceEvent.getFullname)
// 1,                     : Gender        (FaceEvent.getGender)
// 29,                    : Age           (FaceEvent.getAge)
// 0,                     : Smile         (FaceEvent.getSmile)
// 3,                     : Headposes[0]  (FaceEvent.getHeadposes)
// 0,                     : Headposes[1]  (FaceEvent.getHeadposes)
// 3,                     : Headposes[2]  (FaceEvent.getHeadposes)
// 1,                     : PersonId      (FaceEvent.getPersonId)
// 1,                     : Score         (FaceEvent.getScore)
// 2018-10-26 09:52:01, : Date          (FaceEvent.getDate)
// 2018-10-26 09:52:03, : Last          (FaceEvent.getLast)
// 89;                    : Confidence    (FaceEvent.getConfidence)
// 1;                     : Face Size     (VisionEvent.getSize)
// 1;                     : Face Index    (VisionEvent.getIndex)
// 447;                   : Face Rect x   (VisionEvent.getRect()[0])
// 26;                    : Face Rect y   (VisionEvent.getRect()[1])
// 288;                   : Face Rect w   (VisionEvent.getRect()[2])
// 288;                   : Face Rect h   (VisionEvent.getRect()[3])
```

- OnRegisterEvent

```
// event: face registration events
public void OnRegisterEvent(String event)
```

UserData

```
public class UserData {
    public int getPersonId();         //face ID
    public String getNickName();     //face nickname
    public String getFullName();     //face fullname
    public String getEditName();     //face editname
    public String getAge();         //face age
    public String getGender();        //face gender (M : male, F : female)
    public int getScore();        //face score
    public String getDate();        //face register time
    public String getLast();        //face last score time
    public String getHead();        //face head image
    public String getInfo();        //face info
    public String getType();        //face register type
}
```

Summary

1.Stores the face data returned by the vision service.

RobotVisionClient

```
public RobotVisionClient(final String Locker, final Context context, final FaceEventListener
listener,    final boolean withCamera, final boolean withEventlistener)
```

Summary

1.RobotVisionClient constructor.
2.You need to create a RobotVisionClient object before using the vision service functions.
3.The new RobotVisionClient object needs to register the listener of the corresponding function.
4.Please use this in the onCreate method of your Activity.

Parameters

- Locker: The TAG of the current object output log.
- context: The context of the current Activity.
- listener: FaceEventListener, monitors callbacks from RobotVisionClient objects.
- withCamera: the current Activity will open the camera and call the SendFrame interface of the RobotVisionClient.
- withEventlistener: The current Activity listens to RobotVisionClient face recognition events and calls the TurnEvent interface of the RobotVisionClient to turn on face event recognition.

Sample Code

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mRobotVisionClient = new RobotVisionClient(TAG, this, this, true, true);
}
```

Overloading methods

```
@Override
public void onConnectionStatus(boolean isConnected) {
    Log.d(TAG, "onConnectionStatus: " + isConnected);
}

@Override
public void onVisionEvent(String event) {
    Log.d(TAG, "onVisionEvent: " + event);
}

@Override
public void OnRegisterEvent(String event) {
}
```

onResume

```
public void onResume()
```

Summary

1.Notifies the vision service that the Activity is now in the onResume lifecycle state.
2.Please use it in the onResume method of your Activity.

onPause

```
public void onPause()
```

Summary

1.Notifies the vision service that the Activity is now in the onPause lifecycle state.
2.Please use it in the onPause method of your Activity.

onDestroy

```
public void onDestroy()
```

Summary

1.Notifies the vision service that the Activity is now in the onResume lifecycle state.
2.Please use it in the onDestroy method of your Activity.

TurnEvent

```
public void TurnEvent(String event, boolean on)
```

Summary

1.Used to set the state of face events monitored for vision service.
2.Please use it in the onResume method of your Activity.

Parameters

- event: Event type, currently only supports "face".
- on: True: start monitoring face events, False: stop monitoring face events.

SendFrame

```
public void SendFrame(final Context context, final byte[] data, final int width, final int
height)
```

Summary

1.Sends the camera preview to the vision service.
2.Please use it in the onPreviewFrame method of camera.
3.Please set withCamera to true when creating RobotVisionClient.

Parameters

- context: The context of the current Activity.
- data: camera output frame data.
- width: camera output frame data width.
- height: camera output frame data height.

UserAddByPhoto

```
public void UserAddByPhoto(final String locker, final Context context, final String fullname,
String key, final boolean isauto, final boolean saveimage, final String savedir)
```

Summary

1.Call the vision service face registration process: Register via Photo.
2.OnRegisterEvent returns the registration result.

Parameters

- Locker: the TAG of the current object output log.
- context: the context of the current Activity.
- fullname: the full name of the currently registered face.
- key: Registered Face Information Encryption Password.
- isauto：True: photo registration is automatic, False: photo registration is manual.
- saveimage: True: save photo, False: do not save photo.
- savedir: photo save dir

UserAddByVideo

```
public void UserAddByVideo(final String locker, final Context context, final String fullname,
String key, final boolean isauto)
```

Summary

1.Call the vision service face registration process: Register via Video.
2.OnRegisterEvent returns the registration result.

Parameters

- Locker: the TAG of the current object output log.
- context: the context of the current Activity.
- fullname: the full name of the currently registered face.
- key: Registered Face Information Encryption Password.
- isauto: True: video registration is automatic, False: video registration is manual.

UserAddByPortrait

```
public void UserAddByPortrait(final String locker, final Context context, final String
portraitPath, String key, final boolean isauto)
```

Summary

1.Call the vision service face registration process: Register via Portrait.
2.OnRegisterEvent returns the registration result.

Parameters

- Locker: the TAG of the current object output log.
- context: the context of the current Activity.
- portraitPath: the path of the portrait to register.
- key: Registered Face Information Encryption Password.
- Isauto: True: portrait registration is automatic, False: portrait registration is manual.

UserFaceRecognition

```
public void UserFaceRecognition(final String locker, final Context context)
```

Summary 1.Call the vision service face recognition process.

Parameters

- Locker: the TAG of the current object output log.
- context: the context of the current Activity.

UserDelete

```
public boolean UserDelete(final Context context, final UserData user, String key)
```

Summary

1.Calls the vision service face deleting function.
2.Can only remove the faces registered by the current application.
3.Used along with UserGetList.

Parameters

- context: the context of the current Activity.
- user: Face objects to be deleted.
- key: Registered Face Information Encryption Password.

Return

- true: Face data deletion success.
- false: Face data deletion failure.

UserUpdate

```
public boolean UserUpdate(final Context context, final UserData user, final String fullname,
final String age, final String gender, String key)
```

Summary

1.Call vision service face updating function.
2.Can only update the faces registered by the current application.
3.Used along with UserGetList.

Parameters

- context: the context of the current Activity.
- user: face object to be updated.
- fullname: full name of the updated face.
- age: age of the updated face.
- gender: gender (M: male, F: female) of the updated face.
- key: Registered Face Information Encryption Password.

Return

- true: Face data update success.
- false: Face data update failure.

UserGetList

```
public List<UserData> UserGetList(final Context context, String key, final boolean loadhead)
```

Summary

1.Get list of all registered faces.
2.Used with onConnectionStatus.

Parameters

- context: the context of the current Activity.
- key: Registered Face Information Encryption Password.
- loadhead：Whether or not to load the head image.

Return

- List: face data list.

- Overview
- Listen and remove the microphone array to detect the wake-up word reporting event.
- Functional Module
- setMicArrayEventListener

```
AudioManager audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
audioManager.setMicArrayWakeUp(AudioManager.MIC_ARRAY_WU_ENABLE);
// Listen to the microphone array wake-up word event
// Pass parameters: The first parameter is the current application package name, and the second parameter is the wake event callback.
audioManager.setMicArrayEventListener(getPackageName(), new AudioManager.MicArrayEventListener() {
    @Override
    public void onWakeUp(int angle) {
        // Executing a callback indicates that the wake-up event was detected
    }
});
```

removeMicArrayEventListener

```
// Remove the monitor microphone array wake-up word event
// The passed parameter is the current application package name.
```

```
audioManager.removeMicArrayEventListener(String pkgName)
```

Overview

The LifeAssistant class provides an interface for the function of Life Assistant. This document will introduce the variable definitions of the LifeAssistant class and the use as well as parameter description of the API.

Function Module

ACTION_CHANGE_VOICE_STATE

```
private static final String ACTION_CHANGE_VOICE_STATE = "avatar.intent.action.CHANGE_VOICE_STATE";
private static final String EXTRA_VOICE_STATE = "voice_state";
Intent intent = new Intent(ACTION_CHANGE_VOICE_STATE);
// turn on voice prompts
intent.putExtra(EXTRA_VOICE_STATE, true);
// turn off voice prompts
intent.putExtra(EXTRA_VOICE_STATE, false);
sendBroadcast(intent);
```

- **Summary**
  Turn on or off related voice prompts, such as "The network is disconnected", in Life Assistant via Broadcast.
  Note: after turning off the voice prompt of Life Assistant, except that the charging shutdown function relies on voice prompts, Life Assistant no longer gives voice prompts such as wifi status changes.

Overview

RobotSettings provides an interface for robot settings. This section will introduce the definition of variables of the RobotSettings class and the use of APIs and parameter descriptions.

- # Function Module
- ## openSettings

```
public static boolean openSettings(Context context)
```

- **Summary**
  Open the main interface of settings.
- **Return**
  Return true if opened successfully, false otherwise.
- ## openWifiSettings

```
public static boolean openWifiSettings(Context context)
```

- **Summary**
  Open the wifi settings interface.
- **Return**
  Return true if opened successfully, false otherwise.
- ## openBluetoothSettings

```
public static boolean openBluetoothSettings(Context context)
```

- **Summary**
  Open the Bluetooth settings interface
- **Return**
  Return true if opened successfully, false otherwise.
- # Application
- # Personate
- Personate is an application that simulates some human behavior and helps robots interact with people. The main function includes: when in the Idle state, the robot arm performs a slight bending action; when the wake-up word is called, the robot can turn around; after recognising the hand waving, the robot will give a response; the robot can greet the person actively and follow the human face by face recognition.
- # Sample Code
- Personate Sample Code：**Personate**

Overview

All interface types use the http interface method. In the following interface descriptions, the broadcast discovery protocol IP is set to 255.255.255.255, and the rest of the interfaces assume that the robot's LAN IP is 192.168.1.100.

Protocol Interface

BroadCast Discovery Protocol

caller：client program

callee：Avatar Robot

call logic：The client program needs to build a UDP broadcast sending program in advance. After the robot receives the UDP broadcast from the client, it will return the IP address of the robot and the parameters related to the robot.

call method：
The relevant parameters of the broadcast sent by the client are as follows:

- **The port number**：7612
- **broadcast address**：255.255.255.255
- **broadcast content**： "This is a broadcast message sent to a mobile IRemoter when it goes online: find device", this sentence is converted into byte

The client can receive the broadcast content returned by the robot and listen to the port number 7612.

call example：

- **Client sends UDP**

```
class SendTestTask implements Runnable {

    private static final String TAG = "SendTestTask";
    private static final int BROADCAST_PORT = 7612;
    private static final String BROADCAST_HOST = "255.255.255.255";
    private String cmd = "This is a broadcast message sent by a mobile IRemoter when it goes
live: find device";


    @Override
    public void run() {
        byte[] data = cmd.getBytes();
        sendBroadcast(data);
    }

    private void sendBroadcast(byte[] data) {
        DatagramSocket ds = null;
        try {
            //1、Create DatagramSocket for UDP data transfer
            ds = new DatagramSocket();
            LogUtil.d("send_broadcast");
            //2、Create the packet that needs to be sent
            DatagramPacket packet = new DatagramPacket(data, data.length,
                    InetAddress.getByName(BROADCAST_HOST), BROADCAST_PORT);
```

```
            //3、send
            ds.send(packet);
        } catch (UnknownHostException e) {
            Log.e(TAG, "sendBroadcast failed: ", e);
        } catch (SocketException e) {
            Log.e(TAG, "sendBroadcast failed: ", e);
        } catch (IOException e) {
            Log.e(TAG, "sendBroadcast failed: ", e);
        }
    }
}
```

- **Client receive UDP example**

```
class ReceiveTask implements Runnable {
    private static final String TAG = "ReceiveTask";
    static final int port = 7612;


    public void run() {
        //1、create DatagramSocket;
        DatagramSocket socket = null;
        try {
            //2、Create packets for receiving content.
            byte[] buffer = new byte[1024 * 4];

            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

            socket = new DatagramSocket(port);

            while (true) {
                socket.receive(packet);
                String s = new String(buffer, 0, packet.getLength());
                NetEvent.ReceiveData result = new NetEvent.ReceiveData(packet.getAddress()
                        .getHostAddress(), port, s);
            }
        } catch (SocketException e) {
            Log.e(TAG, "ReceiveTask SocketException: ", e);
        } catch (IOException e) {
            Log.e(TAG, "ReceiveTask IOException: ", e);
        } finally {
            if (socket != null) {
                socket.close();
            }
            Log.d(TAG,"ReceiveTask finally");
        }
    }
}
```

Heartbeat Protocol

Http Get request

request address :

```
http://192.168.1.100:8080/?method=heartbeat&&text="iRemoter http update robot info"
```

return value : as json string

Return value example :

```
jsonString={"sex":"0","name":"avatarmind","status":"settings","power":"97","command":"online","sn":"AECHABE18120074","__robotIp":"ip","awake":"1","cid":"0"}
```

Voice Interface

Set the ASR speech recognition content reporting address

caller：client program

callee：Avatar Robot

call logic：The client program needs to set up the reporting server in advance, and then call this interface to set the reporting server address. This interface needs to be called again after the robot restarts.

call example：

```
http://192.168.1.100:8080/?method=setAsrRequestUrl&&url=http://XXX.XXX.XXX.XXX/robot
```

return value：This interface returns a string describing the setting result.

Return value example。

```
{success, url:http://XXX.XXX.XXX.XXX/robot}
```

ASR speech recognition content reporting

caller：Avatar Robot

callee：client program

call logic：Listen to port number 7888, and send the content as a JSON format string: {"asr":"XXXX"}

ASR Switch Interface

call method：

```
http://192.168.1.100:8080/?method=asrSwitch&&param="on"
```

Interface Description：param value - on means on, off means off.

TTS speech synthesis and playback

caller：client program

callee：Avatar Robot

Interface Description：The client program calls this interface to send the text content that needs to be synthesized.

call example：

```
http://192.168.1.100:8080/?method=startSpeak&&text="hello"
```

return value：Returns a Json format data, including an int value named "result" representing the requestId that initiated the voice command.

Return value example

```
{“result”:-1}
```

Cancel voice synthesis playback

caller：client program

callee：Avatar Robot

Interface Description： The client program can call this interface to cancel the current speech synthesis program and cancel the current playing content.

call example：

```
http://192.168.1.100:8080/?method=stopSpeak
```

return value：Returns a Json format data, including a boolean value named "result" representing the execution result.

Return value example

```
{“result”:”true”}
```

Action Interface

Action Interface

caller：client program

callee：Avatar Robot

Interface Description：This interface supports the motion of the robot head and chassis. The input parameters are the head or chassis motion control command param, and the value passed in by the chassis motion.

Notice：When param is the head movement, that is, when the selected parameters are headForward, headBack, headLeft, headRight, the value of the incoming value can be any value, and the robot side will automatically filter out the value of the value. If there is no value, the robot

executes the default value, that is, the default value of the forward and backward distance is 50cm, and the default rotation angle is 90°.

- **Robot Head and Chassis Actions**

call example：

- **has value**

```
http://192.168.1.100:8080/?method=remoteControl&&param="goForward"&&value="1000"
```

- **no value**

```
http://192.168.1.100:8080/?method=remoteControl&&param="goForward"
```

return value：Returns a string describing the command delivery result.

Return value example

```
{remoteControl headForward executed}
```

The optional values of the param parameter are as follows：

name	meaning
headForward	head up
headBack	head down
headLeft	head left
headRight	head to the right
goForward	chassis forward
goBack	Chassis backwards
turnLeft	Chassis left
turnRight	Chassis right

- **Robot internal movement (arm)**

call example：

```
http://192.168.1.100:8080/?method=doAction&&action="WAVE"
```

return value：Returns a string describing the command delivery result.

Return value example

```
{doAction WAVE executed}
```

The optional parameters of action are as follows：

name	meaning
SHAKE_HANDS	shake hands
WAVE	wave
CHEER	cheer
RUN	run
CLAP	applaud
AKIMBO	akimbo
SALUTE	salute
FOLDARM	arms folded
FLYKISS	blow kisses
HIGHFIVE	high five
HUG	Embrace
YES	yes
NO	no
THANK2	thank2

name	meaning
THANK3	thank3
LAUGH	laugh
LISTEN	listen
LOOK1	look1
LOOK2	look12
WORRY	worry
SHY	shy
TELL	tell
ME	me
WE	we
HANDSBACK	handsback
WAKE	wake
TICKLE_RIGHT	itching on the right
TICKLE_LEFT	left itching
PUSH_FORWARD	advance
INCOMING_CALL	call
MJ_DANCE_POSE	MJ classic dance moves
DANCE_POSE	classic ballet moves
TAKE_PHOTO	takephoto
UPGRADE	upgrade

name	meaning
TURN_BOOK	flip book
STOP_TALKING	forbidden to speak
DON_TTOUCH_ME	Do not touch me
OH_YEAH	Oyer
TO_FOLLOW	ready to follow
FOLLOWING	following
WIPE_PERSPIRATION	wipe sweat
RAIN	rain
SNOW	snow
SELF_PROTECTION	self protection
NARRATE	Tell the story of Sinology
IDLE	state of nature
CHAT_1	chat1
CHAT_2	chat2
CHAT_3	chat3
CHAT-4	chat4
PLEASE	please

Action Stop Interface

caller：client program

callee：Avatar Robot

Interface Description：This interface stops all movements the robot is running, including chassis and head movements.

- **Robot head and chassis motion stopped**

call example：

```
http://192.168.1.100:8080/?method=stopAction
```

Listener Callback for Completion of the Action

caller：client program

callee：Avatar Robot

interface logic：After the robot performs the action, it will send a UDP broadcast, and the client can listen to the port number 7900. The broadcast content is a json string of type {String,boolean}. The String type can be "wheelOrHead" or "otherAction", "wheelOrHead" represents the execution of head or chassis motion, and "otherAction" represents the execution of arm motion. When the boolean type variable is true, it means that the action has been executed, otherwise, it has not been executed.

Motor Control Interface

caller：client program

callee：Avatar Robot

Interface Description：This interface supports the client to control the motors of the relevant parts of the robot. param is the motor position, and value is the motor angle. The meaning of the motor position and the corresponding angle range are described below.

name	meaning	scope(°)
ARM_ROTATION_RIGHT	right arm rotation	[-25, 175]
ARM_SWING_RIGHT	right arm swing	[0, 65]
FOREARM_ROTATION_RIGHT	Right forearm rotation	[-80, 80]
FOREARM_SWING_RIGHT	Right forearm swing	[0, 90]
WRIST_RIGHT	right wrist rotation	[-80, 80]
ARM_ROTATION_LEFT	left arm rotation	[-25, 175]

name	meaning	scope(°)
ARM_SWING_LEFT	left arm swing	[0, 65]
FOREARM_ROTATION_LEFT	Left forearm rotation	[-80, 80]
FOREARM_SWING_LEFT	Left forearm swing	[0, 90]
WRIST_LEFT	left wrist rotation	[-80, 80]
NECK_ROTATION	neck swing	[-50, 50]
NECK_TILT	neck swing up and down	[-10. 28]

call example：

```
http://192.168.1.100:8080/?method=motorsControl&&param="ARM_ROTATION_LEFT"&&value="10"
```

return value：Returns a string describing the command delivery result.

Emoji Interface

caller：client program

callee：Avatar Robot

Interface Description：The interface for the robot to execute the expression, the values that can be passed in param are shown in the following list.

name	meaning
CLEAR	clear all emoticons
SMILE	Smile
SAD	sad
LAUGH	laugh
SURPRISE	surprise
CRY	cary

name	meaning
DOUBT	doubt
SHH	shh
SHY	shy
COVER_SMILE	hide sadness with laughter
GRIMACE	grimace
NAUGHTY	naughty
HEARTED	enthusiastic
ANGRY	angry
THINKING	thinking
POWER_ON	Charge
POWER_OFF	shutdown
WAKE_UP	wake
SLEEP	hibernate
TALK	dialogue
LISTEN	listen
DEFAULT	default state
BLINK	wink
EYECLOSE	eyes closed
EYEOPEN	open eyes

name	meaning
FROWN	frown
EYEBINDONE	first sight
INDIFFERENT	indifferent

call example：

```
http://192.168.1.100:8080/?method=emoji&&param="LAUGH"
```

Sensor Interface

caller：client program

callee：Avatar Robot

Interface Description：Robot-related sensor events will be sent through UDP broadcast, the port number is 7888, and the client program can listen to port 7888. The sent content is a string in JSON format: {"type": "sensorEvent"}, and the related values of sensorEvent are shown in the following table.

name	meaning
RC_FRONT_UPPER_OBSTACLE	Obstacle in front
RC_FRONT_LOWER_OBSTACLE	Obstruction in the lower part of the front
RC_BACK_UPPER_OBSTACLE	Obstruction in upper rear
RC_BACK_LOWER_OBSTACLE	Obstruction in the lower rear
RC_LEFT_OBSTACLE	Obstruction on the left
RC_RIGHT_OBSTACLE	Obstruction on the right
RF_HEAD_TOUCH	head
RF_LEFT_SHOULDER_TOUCH	left arm
RF_RIGHT_SHOULDER_TOUCH	right arm

name	meaning
RF_LEFT_OXTER_TOUCH	left armpit
RF_RIGHT_OXTER_TOUCH	right armpit

Note Playback Interface

caller：client program

callee：Avatar Robot

Interface Description：The interface for the robot to perform note playback. The values that can be passed in param are as follows:

name	meaning
1	Alto 1
2	Alto2
3	Alto3
4	Alto4
5	Alto5
6	Alto6
7	Alto7
8	Treble 1
9	Treble 2

name	meaning
10	Treble 3
15	bass1
16	bass2
17	bass3

call example：

```
http://192.168.1.100:8080/?method=playNotes&&param="1"
```

Screen Display Interface

Screen Display Interface

caller：client program

callee：Avatar Robot

Interface Description：This interface supports the function of scrolling text displayed on the robot screen, and param is the text content to be displayed.

call example：

```
http://192.168.1.100:8080/?method=board&&param="hello"
```

return value：Returns a string describing the command delivery result.

The screen shows the stop interface

caller：client program

callee：Avatar Robot

Interface Description：This interface stops the screen display function of the robot.

call example：

```
http://192.168.1.100:8080/?method=boardClose
```

Gyro

How to get started :

```
http://192.168.1.100:8080/?method=gyroStart
```

end method :

```
http://192.168.1.100:8080/?method=gyroStop
```

return value : Just listen to the port number 7888, the data format is json format, the example is as follows.

```
{"gyroscope":"{"gyroscopeX":29.932395935058594,"gyroscopeY":101.70033264160156,"gyroscopeZ":-21.066123962402344}"}
```

Player Interface

How to get started :

```
http://192.168.1.100:8080/?method=avatarPlayerStart&&param="fileName"
```

Interface Description : The param parameter is the name of the arc file to be played. The arc file needs to be pushed to the path of `sdcard/media/` in advance.
end method :

```
http://192.168.1.100:8080/?method=avatarPlayerStop
```

Audio File Playback

How to get started :

```
http://192.168.1.100:8080/?method=playHandyMusic&&param="fileName"
```

Interface Description : The param parameter is the name of the audio file to be played (full name, such as xx.mp3). The file needs to be pushed to the path of `sdcard/HandyBlock/` in advance.

Decibel Measurement Interface

Decibel measurement on

call method :

```
http://192.168.1.100:8080/?method=decibelStart
```

Interface Description：Broadcast interface 7888, json key value decibel

Decibel off

call method：

```
http://192.168.1.100:8080/?method=decibelStop
```

Introduction

Avatar Studio

Avatar Studio is a tool developed by AvatarMind Technology for editing robot motion and robot media resources. The download is available **here**.

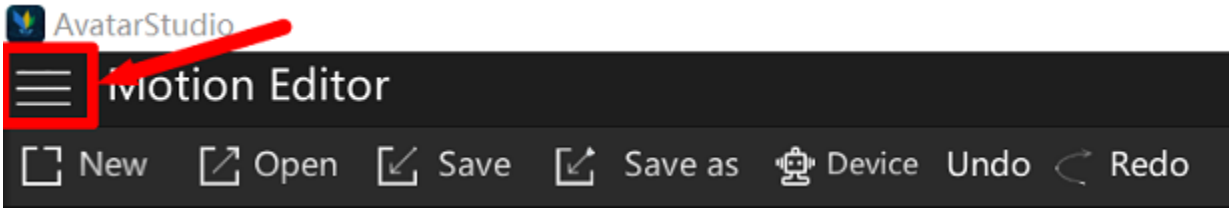
iPal is a smart robot developed by AvatarMind Technology and serves as the model in Avatar Studio.

Each of iPal's movable joints is operated by a motor that users can set to specified angles. Avatar Studio allows users to take control of iPal's motors to create compound actions.

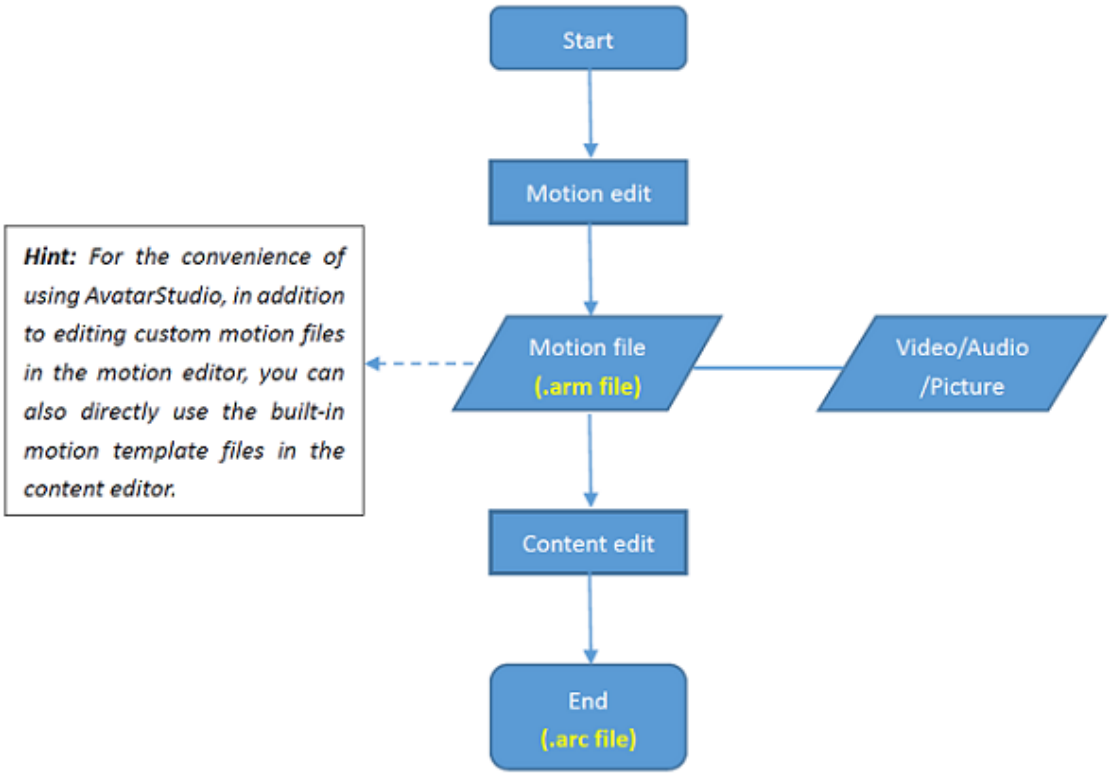
Avatar Studio also provides a robot simulator to simulate movement files without a robot.

Functionality

The two key components of Avatar Studio are **Motion Editor** and **Content Editor**. They can be found by clicking the menu icon in the top left corner.



Usage flow

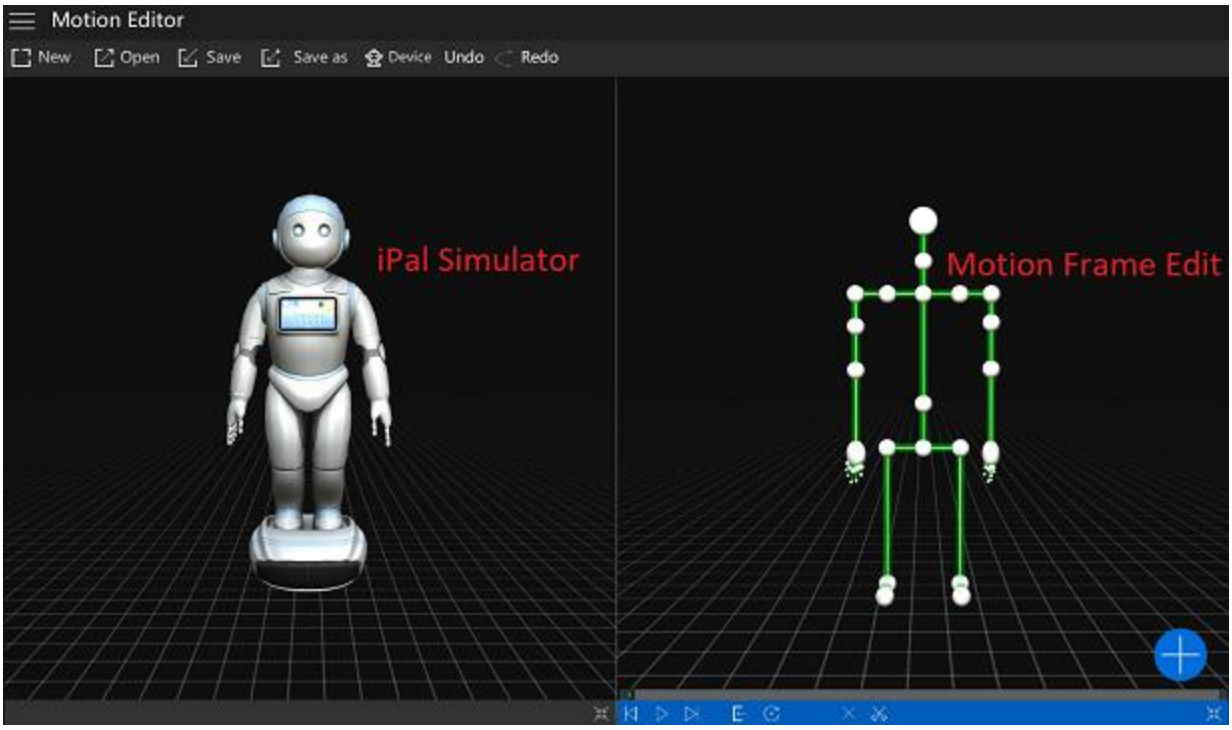


Motion Editor

Motion Editor has 2 panels:

- iPal Simulator
- Motion Frame Edit

Motion Editor creates `.arm` files that can be played on iPal.

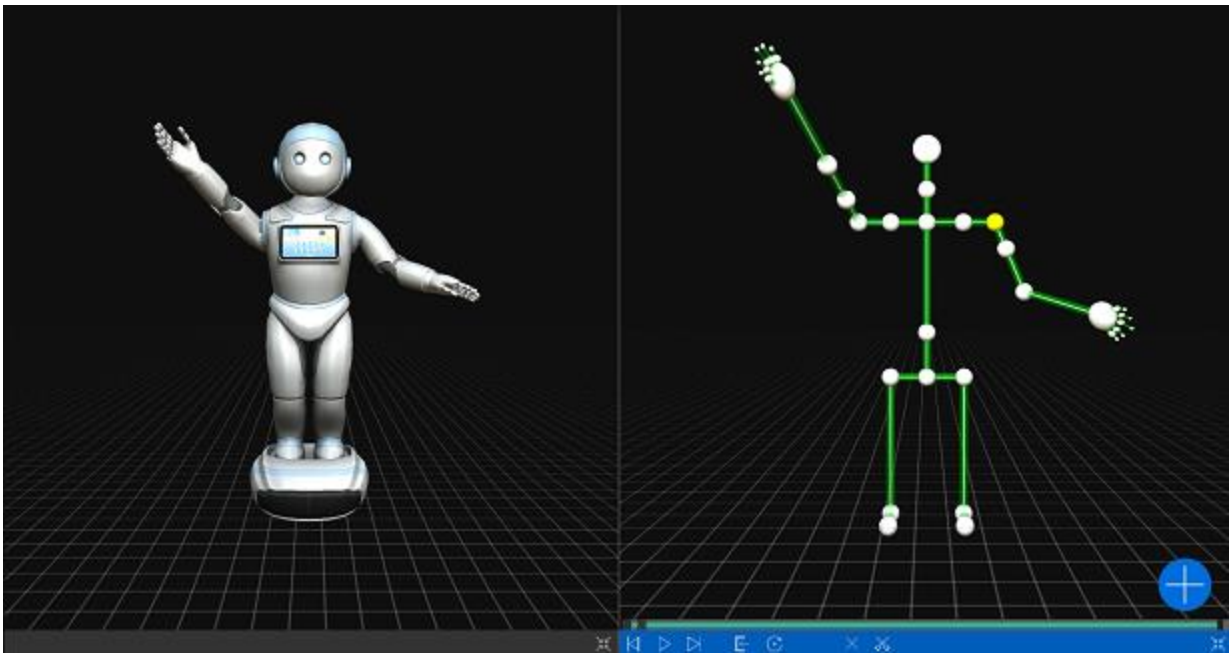


Hint: Each panel can be clicked to display their corresponding setting details on the right side of the screen.

Motion Frame Edit

Overview

Custom motions can be edited here. It shows the Avatar skeleton which represents all of the joint motors on an iPal robot. Clicking and dragging a white sphere allows control of each joint. The iPal simulator on the left will move along with the skeleton, allowing the user to observe how a real robot would act. The Motion Frame Edit also shows positions. Each position counts as a frame. Motor positions can be defined during simultaneous or unique frames, and a series of frames create an .arm file. Here is an example as shown in below.



Menu

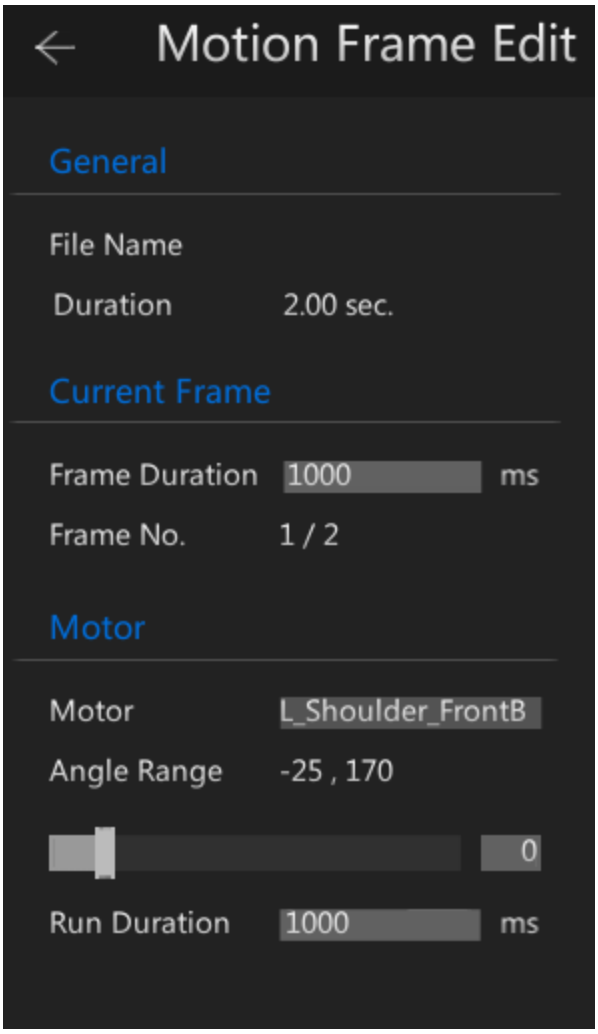
When editing the motion frame, the .arm file can be played at any time. The playing menu is shown below.

From left to right:

- **Previous frame**
- **Pause/Resume**
- **Next frame**
- **Append:** Connect a new .arm file to the end of the current frame
- **Reset:** Reset the current frame
- **Delete:** Delete the current frame
- **Cut:** Removes a portion of an .arm file. Drag along the progress bar to select the portion to be removed.
- **New**  : Add a new frame that is same as the previous frame

Settings

Click the Motion Frame Edit panel. The Settings menu will appear at the right side of the screen as shown below.



- **General**
  It shows the total duration of the current motion file. If a saved .arm file is opened in Avatar Studio, the file name will also be displayed.

- **Current Frame**
  This section will show the details for the current frame, which contain both the total frame count and the position of the current frame within the file. This section can be edited.

    - **Frame duration:** set and modify the duration of a single motion. The default duration is 1000ms, which is 1 second.

      Hint: you can change the motion rhythm by increasing or decreasing the frame duration.

    - **Frame No.:** shows the position of the current frame out of the total frame count.

- **Motor**
  When a motor is clicked on the Motion Frame Edit, this panel will show the details of the motor.
    - **Motor:** the selected motor. It can be selected on Motion Frame Edit or from the drop-down menu.
    - **Angle range:** each motor has a maximum angle range of rotation. An angle can be selected by dragging the slider below.
    - **Run duration:** time required for the motor to move to the target angle.

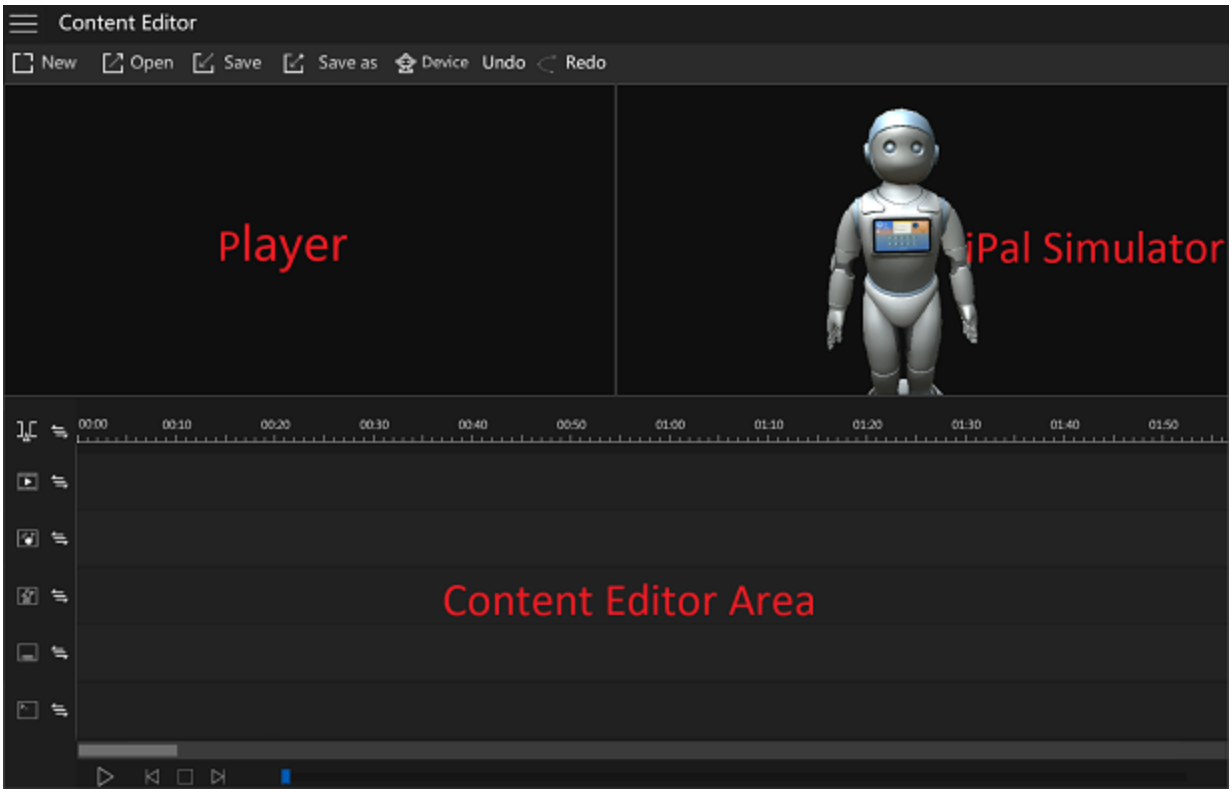
iPal Simulator

The iPal simulator will move along with the skeleton, allowing the user to observe how a real robot would act. Clicking iPal simulator panel to show the role settings of iPal, which can be set to Girl or Boy. The .arc file format combines various types of media resources with an .arm file to form a complete iPal content file.

Content Editor

.arm files can only contain movement sequences. Media resources such as pictures, music, videos, speech, and emoji can be inserted into the sequence, but the resulting file will be an `.arc` file.
Content Editor has 3 panels:

- Player
- iPal Simulator
- Content Editor Area



Player

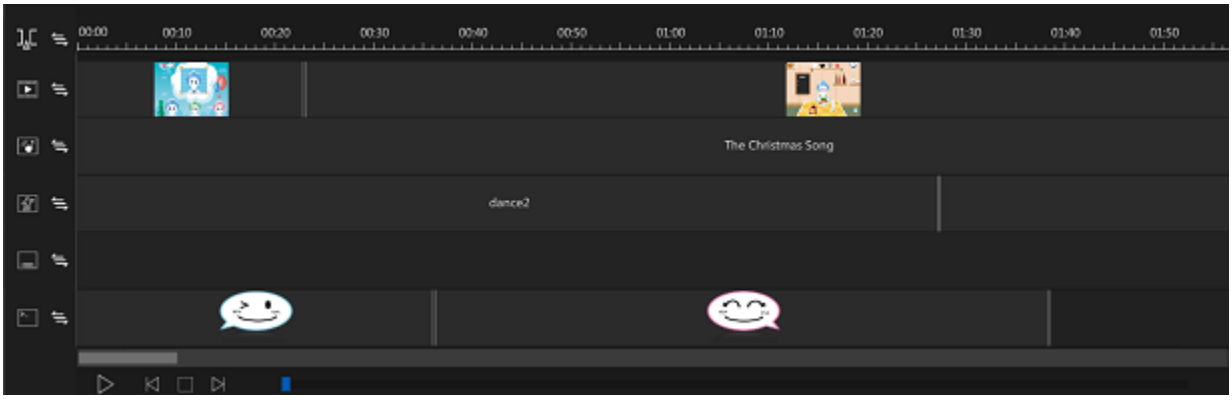
User can preview the pictures or videos added to the content editor on the Player. The Player in Avatar Studio simulates what will be displayed on the robot's chest.

iPal Simulator

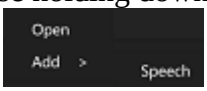
The iPal Simulator is used to play an added .arm file.

Content Editor Area

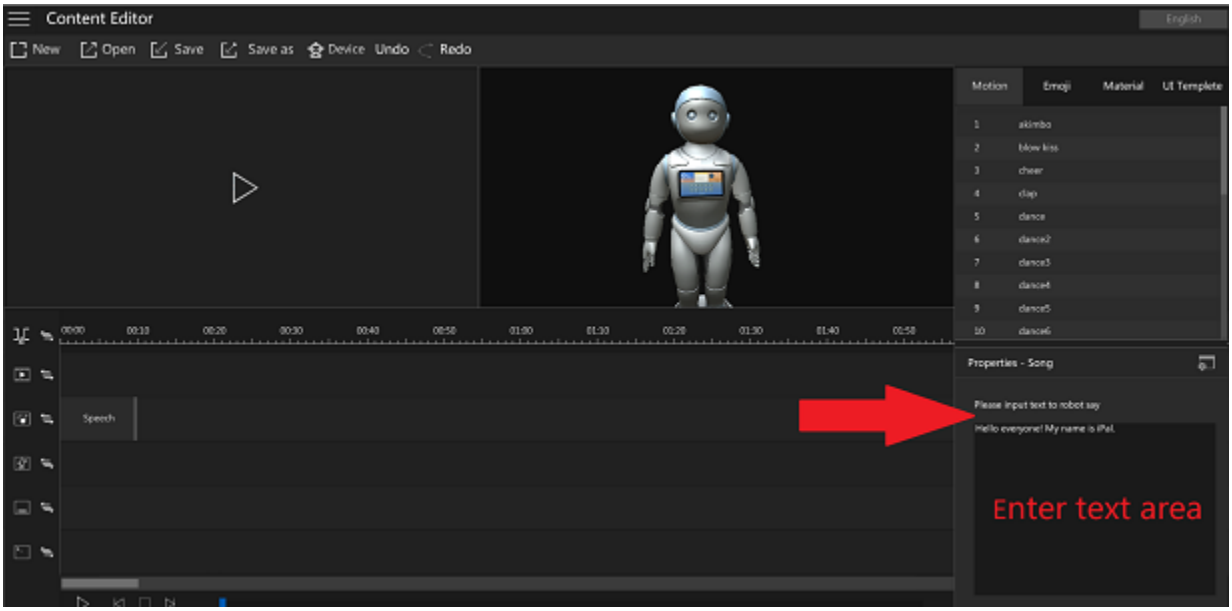
User can add media resources and .arm files into the content editor area as shown below.



The time axis above shows the length of the current .arc file. The resource file's width is equivalent to its duration. The left toolbar shows resource types which may be added to the .arc file.

- To add picture or video click  .
- To add music click  . If you want to add robot voice here, please holding down the left mouse button in the black blank, and then click “Add > Speech” 

to enter the text in the Properties on the right side of the screen, as shown below. As a result, the robot will convert these texts into speech.



Note: It should be noted that Avatar Studio cannot automatically identify the duration of the Speech. After entering the text, please extend the duration of the speech in the time axis as long as possible, then save the current content and transfer it to the robot for playback, and then confirm the exact duration of the Speech. Finally, return to Avatar Studio and drag the Speech to the target time.

- To add the .arm File click  or directly drag the motion template file in the Motion list at the top right of the screen.
- To add emoji, please directly drag the emoji template file in the Emoji list at the top right of the screen.

When you are finished editing, you can save your content as an `.arc` file.

Hint: To modify the start time of multiple types of files simultaneously, please click  on the left toolbar at the same time, and then drag the file with the earlier start time. For example, if

you need to delay the playback of picture and music synchronously (assuming that the start time of the picture is earlier than the start time of the music), you can first click  in front of the picture and music axis, and then drag the picture to the target time to change the starting time of the pictures and music synchronously.

It should be noted that after completing the modification, please cancel the selected state of all the clicked  in time.

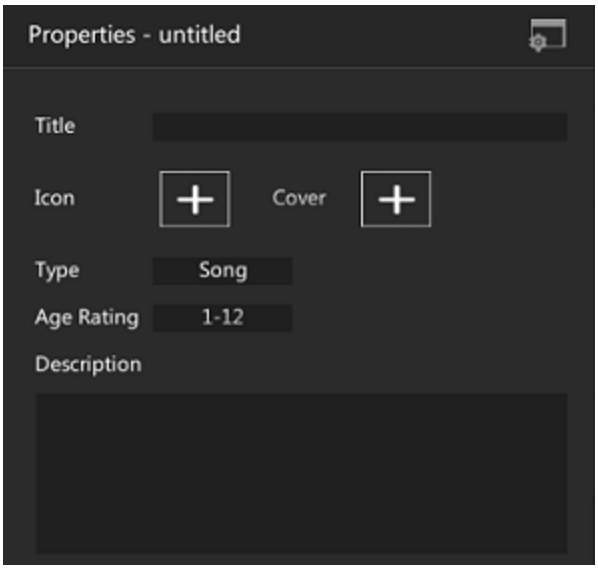
Resource Library

The resource library is in upper right part of the screen. Some motion, emoji, music and video template files are pre-installed in it.

These files can be dragged into the content editor area directly. Avatar Studio will automatically place the file in its respective category.

Save File

Before you prepare to save your content, please fill in the basic information of the file in the Properties on the right side of the screen, as shown in the figure below.



If the information filling box does not appear, please click  and it will appear. Options for filling box are shown below:

- **Title:** file title when viewing this file on robot
- **Icon:** file icon when viewing this file on robot
- **Cover:** picture of first frame when playing on robot
- **Type:** file type, this determines which folder the file will appear in on the robot
- **Age Rating:** age of the intended audience
- **Introduction (optional):** a short description of the file

After completing the basic information of the file, the `.arc` file can be saved.

Previous:be without

Next:iRemoter Manual

Introduction

AvatarMind's iRemoter can be used to remotely control one or more iPal robots which are connected to the same local WiFi network. Using iRemoter you can move iPal around, start and stop songs and stories, change expressions, make poses, speak phrases, and much more. If multiple iPals are connected to iRemoter at the same time, they will be controlled simultaneously. You can even create a synchronized dance activity.

Here is an **example**

Requirements

To use iRemoter you will need a basic Android tablet or phone. A tablet with its larger screen is preferred.

The most important thing to remember when using iRemoter is that iRemoter and the iPal it controls **MUST** be on the same local WiFi network. Most of the time this is the problem if iRemoter does not seem to work. A MiFi, iPhone hotspot, or a local WiFi network in your home or building, should work.

Note: In some cases broad area WiFi networks will not work well. But local network such as a MiFi, iPhone hotspot, or a local WiFi network in your home should work.

Installing iRemoter

The first step is to install the iRemoter apk on your android tablet or phone. Download from this **link**.

Hint: please change browser or unblock browser firewall if you cannot open it.

The iRemoter apk is not on the Google Play store, so you will have to use other methods to install. Four methods are listed below.

using Android devices

- Click the download link provided by AvatarMind directly from the browser on your phone or tablet to download and install iRemoter.apk automatically.
- Go to the QR code on iPal and scan. On iPal the QR code can be accessed by APP → swipe left and choose settings → robot settings → scroll down and select iRemoter.

using Windows PC

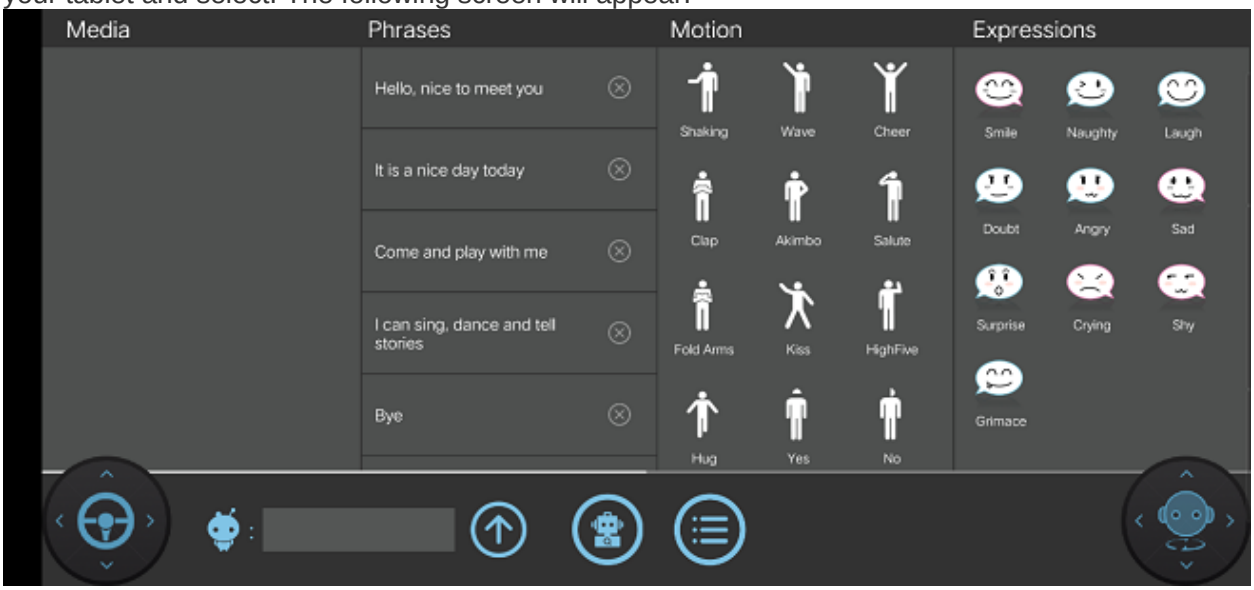
After downloading iRemoter.apk with the Windows PC, **you will need to connect a USB charger cable between your Windows PC and your tablet.** In some cases you may also need to enable USB debugging on your tablet. See the appendix and/or search the internet for instructions for how to do this for your device.

- **using 3rd party tool**
  Use a 3rd party tool to install. One that we suggest is MobileGo. **Versions** are available for both Windows and the Mac.
- **using adb**
  Install using the adb tool in the command prompt window of a Windows PC as
  `C:>adb install iRemoter.apk`

Note: instead of just using “iRemoter.apk” in the above adb command, in general you will have to supply the full path to where iRemoter.apk is located on you PC.

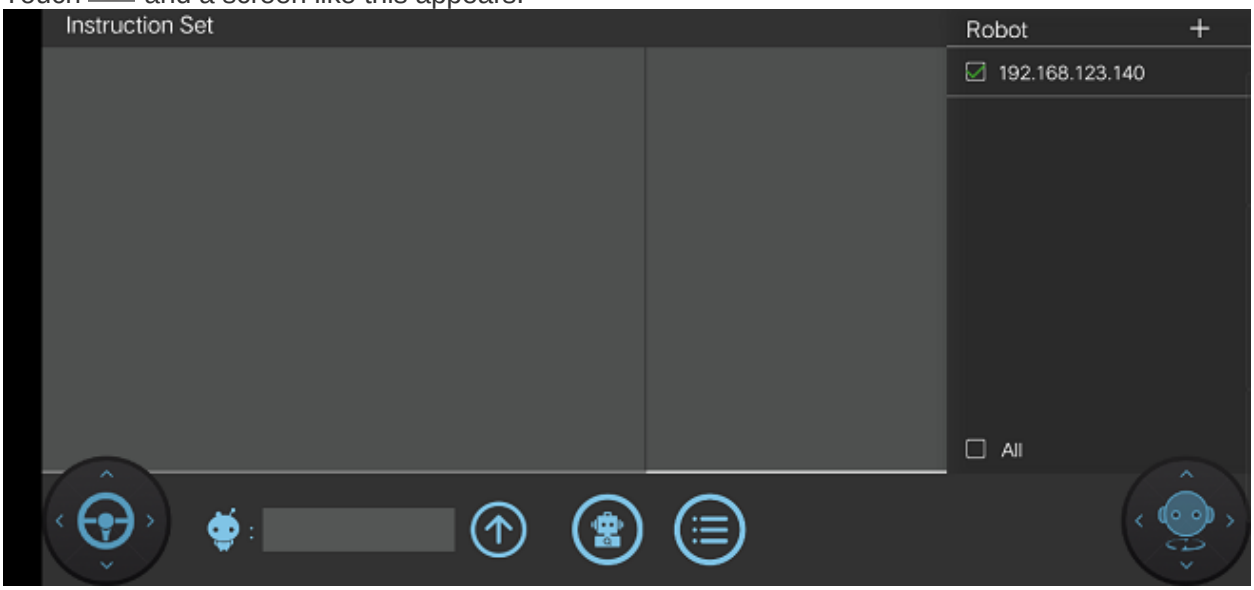
Getting Started

Now that it is installed on your device, you can start iRemoter. Look for its icon in the Apps section of your tablet and select. The following screen will appear.



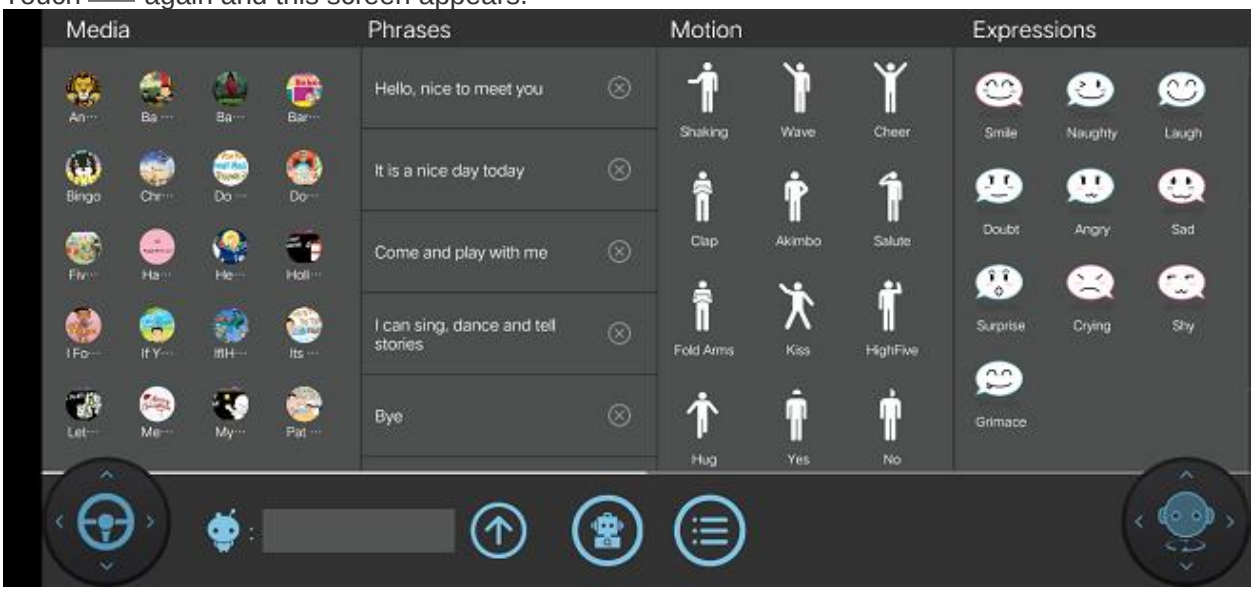
As a Reminder, make sure your iPal and tablet are connected to the same local WiFi network. This is critical.

Touch  and a screen like this appears.



This shows a list of the robots that iRemoter finds on the local network. Click the boxes for all the robots you want to control. If you have only one iPal you will see only one entry. Click the box for that entry. If no robots appear then either the robots are not on the same local WiFi network as the tablet, or the WiFi network is too broad to support the discovery process.

Touch  again and this screen appears.



If your tablet has connected successfully to iPal you will see songs and stories appear in the left-hand panel. If you don't see them, go back and check that the tablet and iPal are on the same local WiFi network.

In all the columns you can scroll down to find additional options and content by swiping up or down.

Using iRemoter

Once you see the screen above the rest is easy and intuitive.

To play a **song** or **story** imply touch its icon. To stop just touch the icon again.

To make iPal **pose** just touch one of the motion in the 3rd column.
Note:Let one pose finish before starting another.

To make iPal change **expressions** just choose one in the Expression column.

You can swipe up or down to see additional items in each column.

To move iPal hold down  at the bottom left with your finger and move up and down, or right and left.
Note: iPal moves best on a hard surface or short carpet.

To move iPal's head do the same thing using  at the bottom right.

To speak a phrase simply touch one of the phrases in the second to left column.

- To add a phrase touch in the box to the right of  . A keyboard will appear and you can enter a new phrase. You can enter a phrase by either typing it in on the keyboard or by speaking it (there is a microphone symbol  at the top of the keyboard you can press to start speaking). When finished activate the phrase by touching  .

- To delete a phrase touch .
- To change the order of the phrases put your finger on a phrase and move it up or down until you find the new position you want and release. If you switch on the button

in the , iPal will speak the phrases you touch one after the other in sequence.

To adjust the volume touch and then click

.

To make and record your own command that can control the robot based on your ideas, please touch and click

- Once you start recording you will see red symbols appear at the top of the screen. One shows the recording time and is to stop recording.

- When recording is finished, you will be asked to enter a name and a command file will be generated and saved to the Instruction Set column which is to the right of the expression column (to access swipe left on the screen). Then to activate the instruction by clicking the command file name.
- To delete a command file touch .

Additional Comments

Commands can be given at the same time to get simultaneous or near simultaneous actions. For example, to make the robot pose and change expression you touch two icons in rapid succession. If you use one hand had to move iPal using then with the other hand you can choose motions, expressions, and songs while iPal is moving.

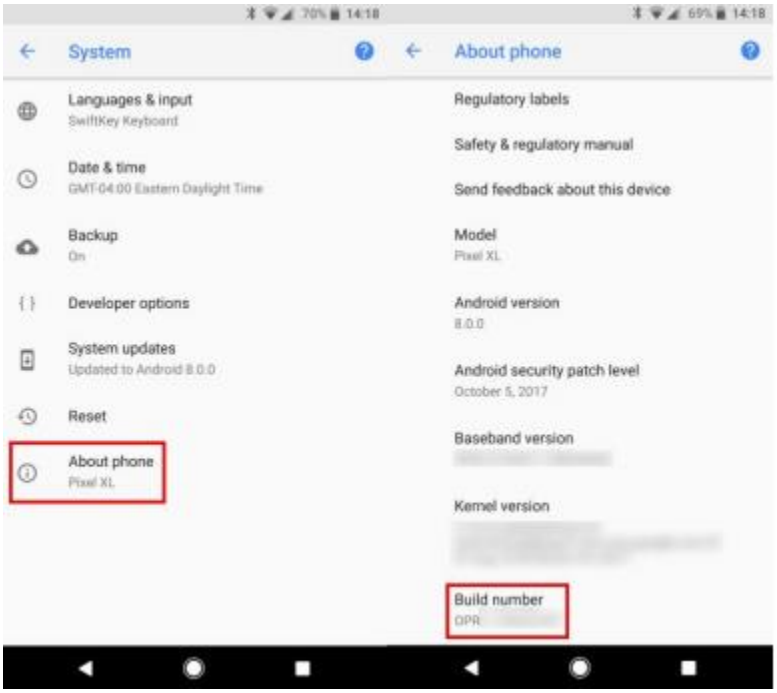
Appendix

This part will introduce how to enable USB debugging on an Android tablet.

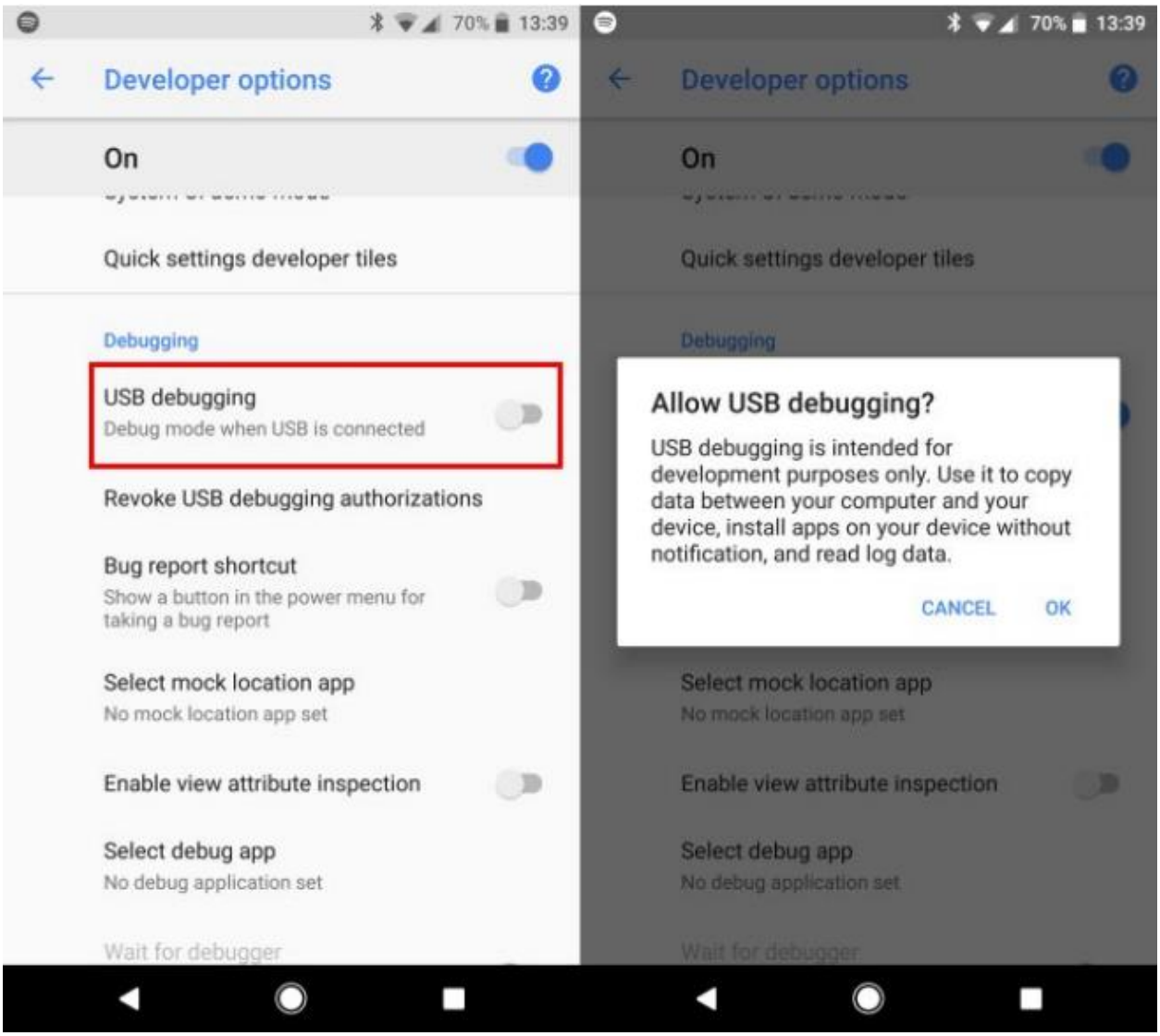
The following procedure will work for most recent tablets. But if it does not work for your tablet then just search on the internet for how to do it for your particular tablet.

On modern Android devices, you'll find USB Debugging in the Developer Options menu, which is hidden by default. To unlock it, go to **Settings** and scroll down to until you see **About** and then

select. Then scroll down in the **About** section until you see a **Build number** entry. Tap it 5-6 times and you'll see a notification letting you know that you're now a developer.



Jump back to **Settings**, and scroll back down to the bottom where **About phone** is. You'll see a new entry, **Developer options**. Tap this, and look for **USB debugging** under the **Debugging** header. Hit the slider to enable it (in the picture below it is off) and confirm Android's warning that you understand what this feature is for.



iPalHelper

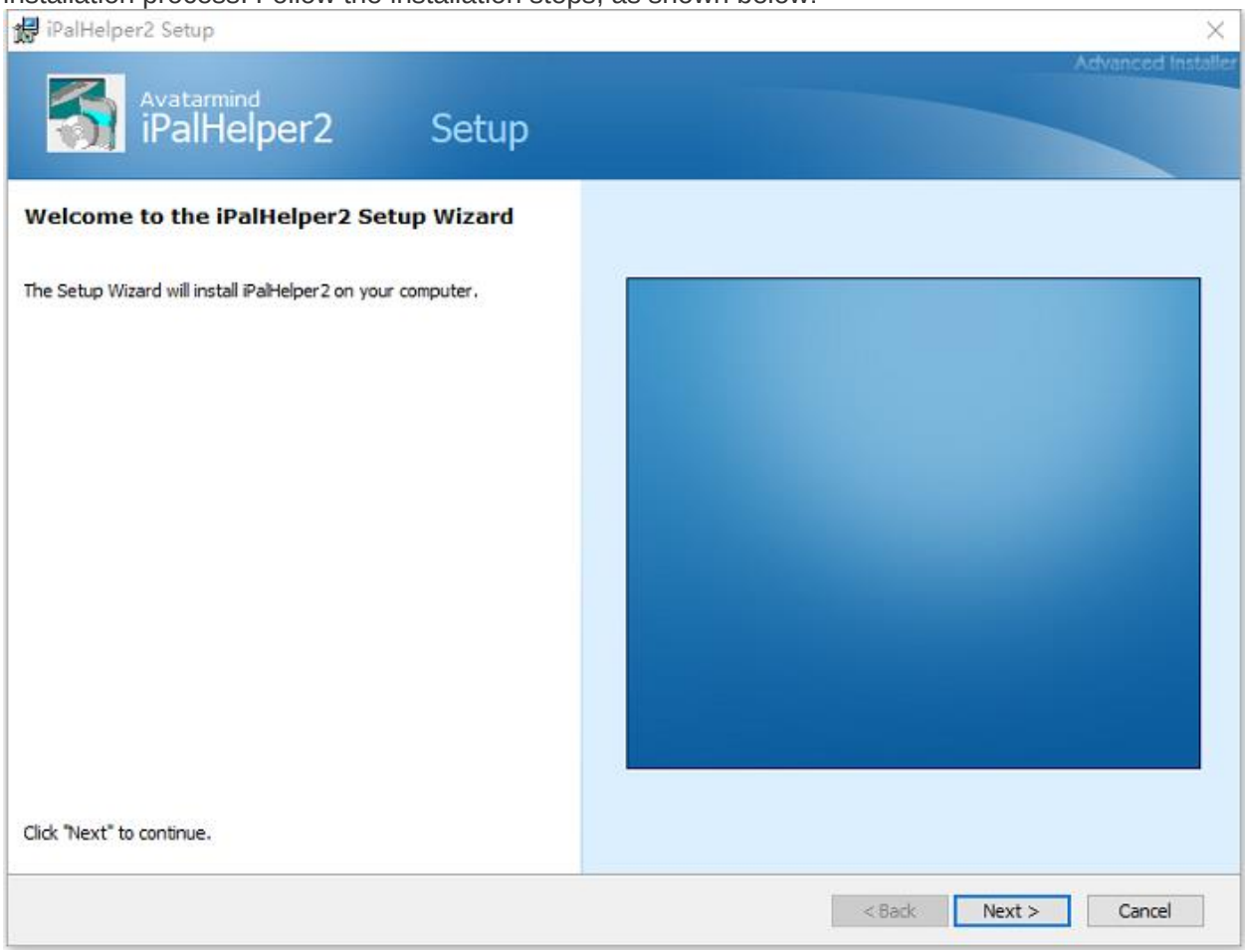
Avatar Robot Helper, i.e. iPalHelper, is developed by AvatarMind and is used to manage robot resources. You can upload local files, delete data or resources and update the robot's operating system through iPalHelper. Testers can use iPalHelper to catch log files while the robot is running. The download is available here: **iPalHelper**

Getting Started

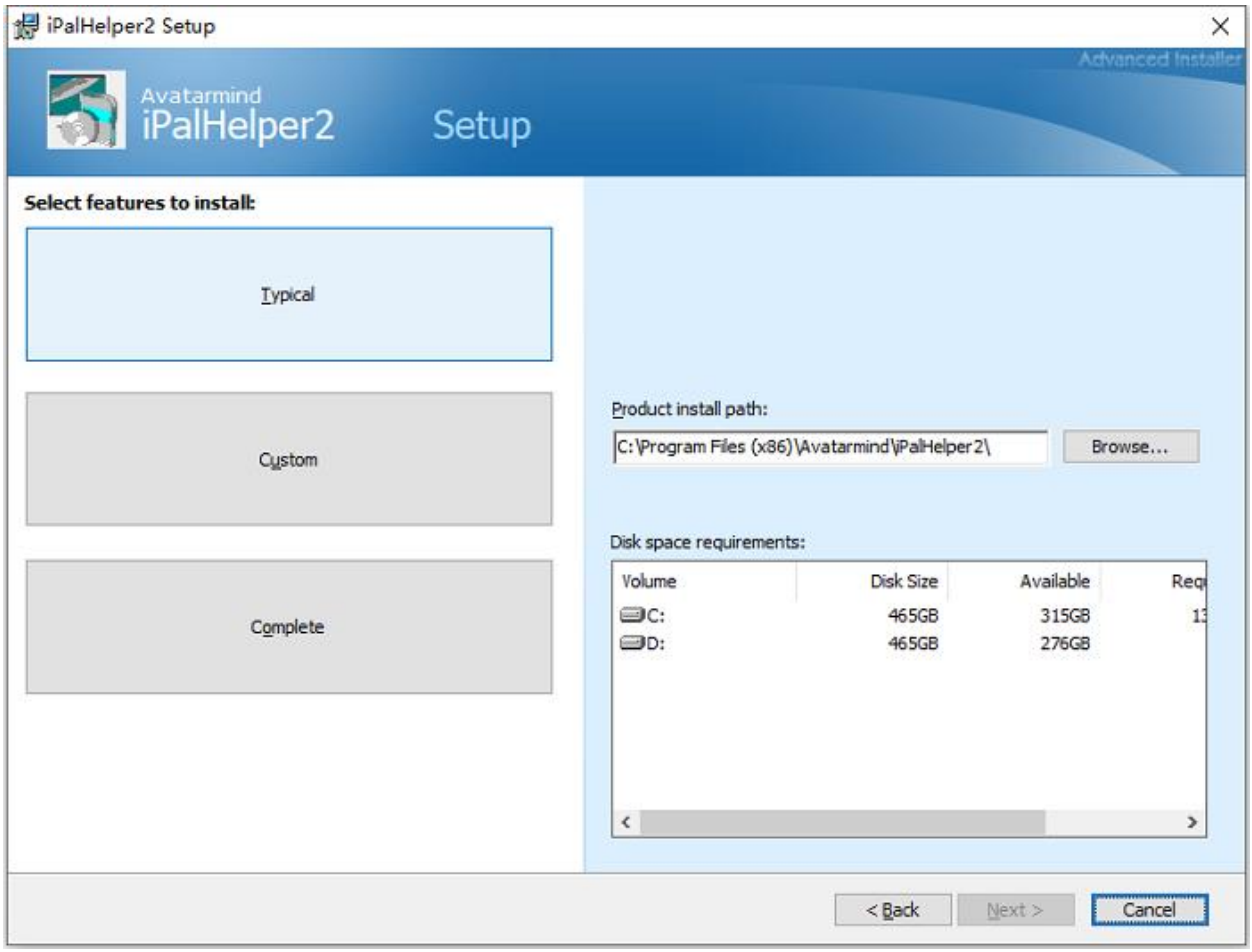
Setup

- **System requirements**
  System: Windows7, Windows10; 32 or 64 bit
  Hardware: requires a USB 3.0 port

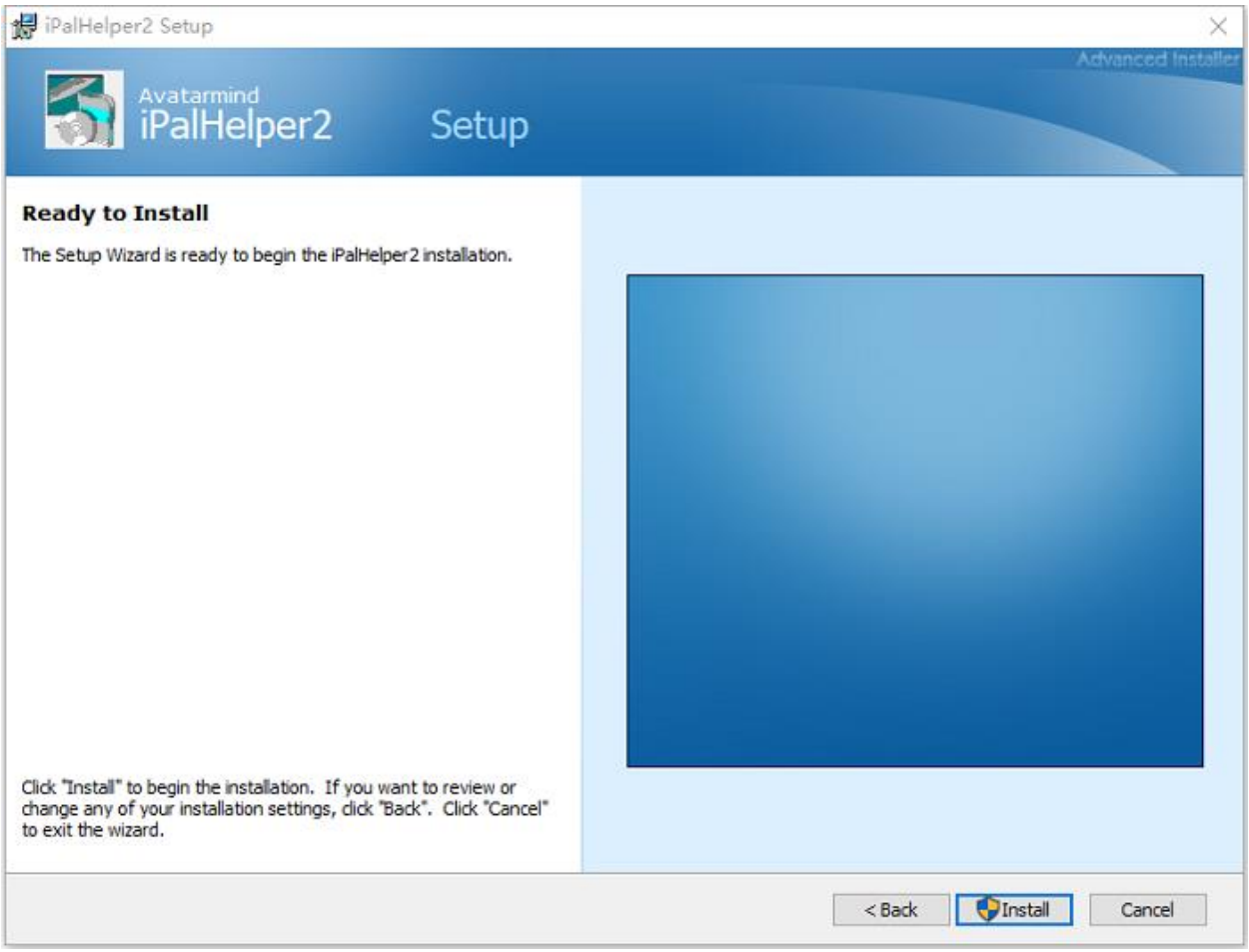
- **Setup process**
  Load **iPalHelper2_setup_v2.0.0.msi** onto your PC and then double click to start the installation process. Follow the installation steps, as shown below:



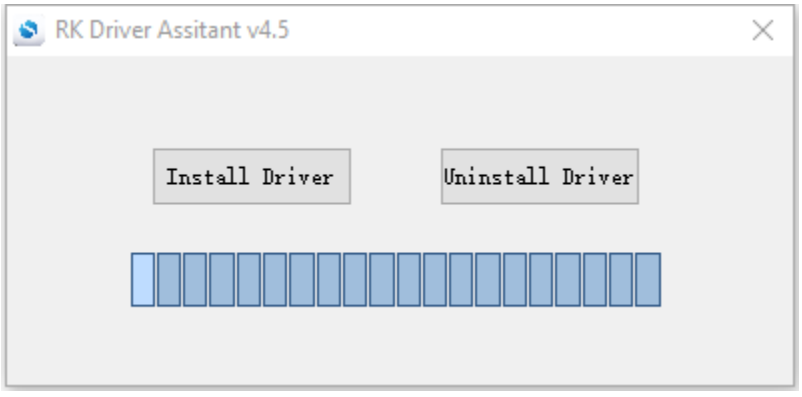
  Choose **Typical** mode.



Continue follow the installation steps.

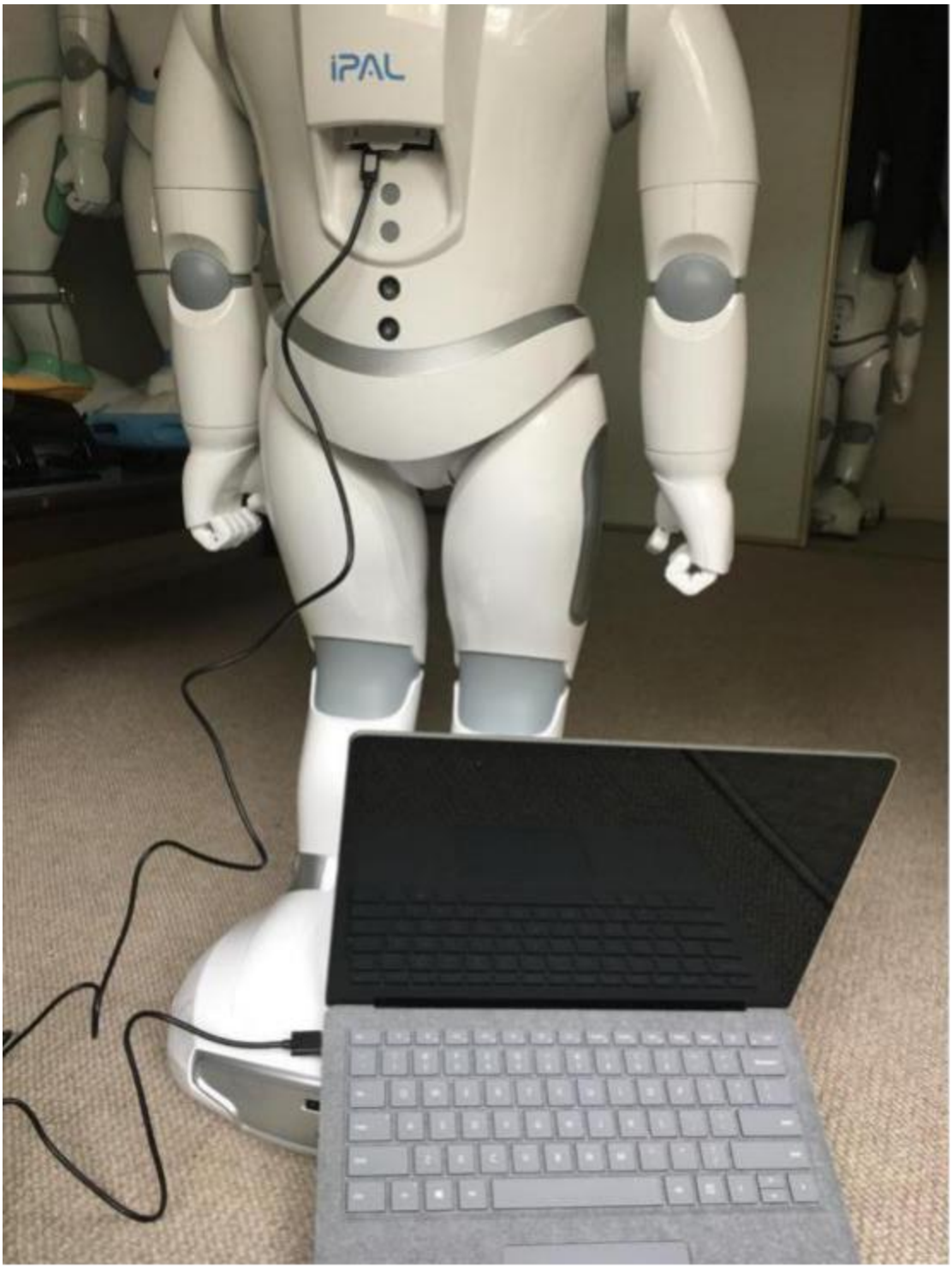


If prompted to install the Driver shown below, please install it, otherwise the computer may not be able to detect to the robot.



Run iPalHelper

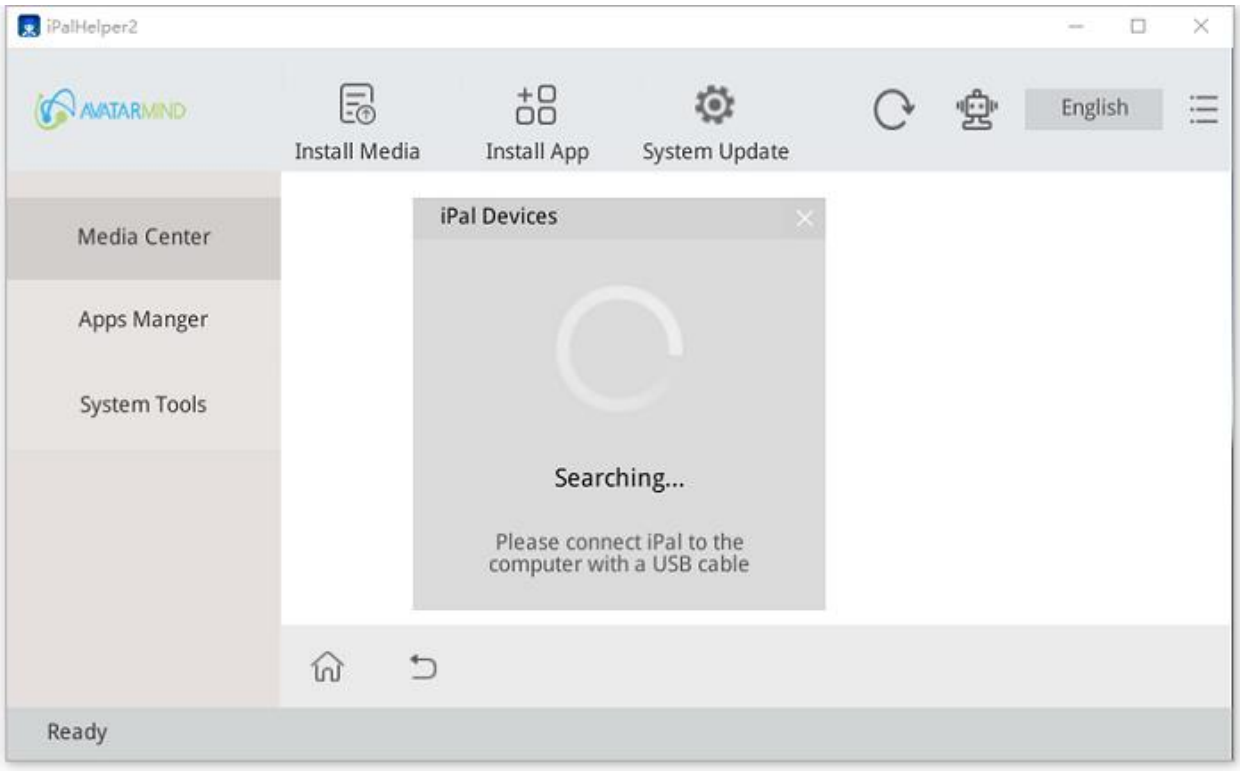
After the installation is complete, the iPalHelper icon will appear on the desktop or start menu. Please connect the iPal® robot by a USB type-C cable to your PC. See picture below.



- **Search for iPal devices**

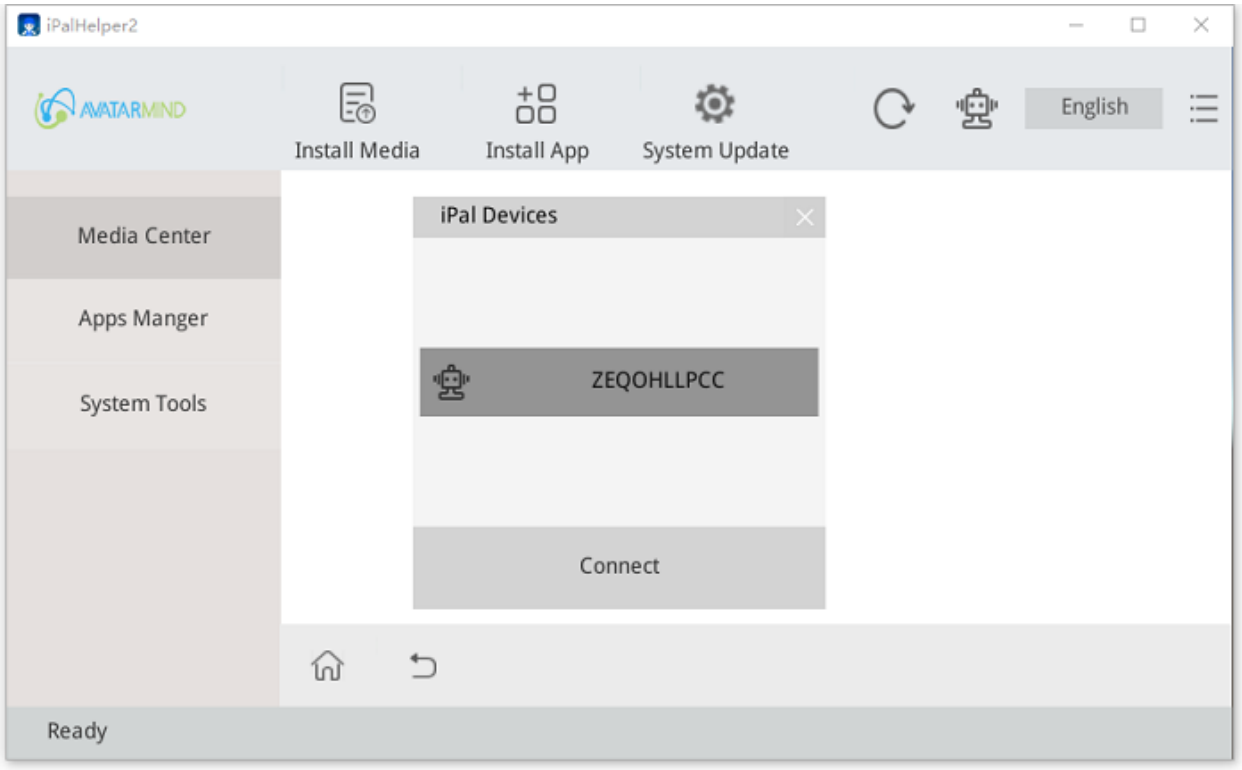
  After double clicking to start iPalHelper, it will search for connected iPal devices after launching. If iPalHelper does not find any device (iPal), please check that the PC and iPal are connected with a USB cable. If there is still no detection of iPal, try to reinstall the driver

program. The path of driver program is where iPalHelper is installed plus **tools\USBDriver\DriverInstall.exe**.



- **Connecting iPal**

  The iPal detected will be shown below Click to connect. For the 1st time connection, iPalHelper will make necessary configuration changes so it may take longer.

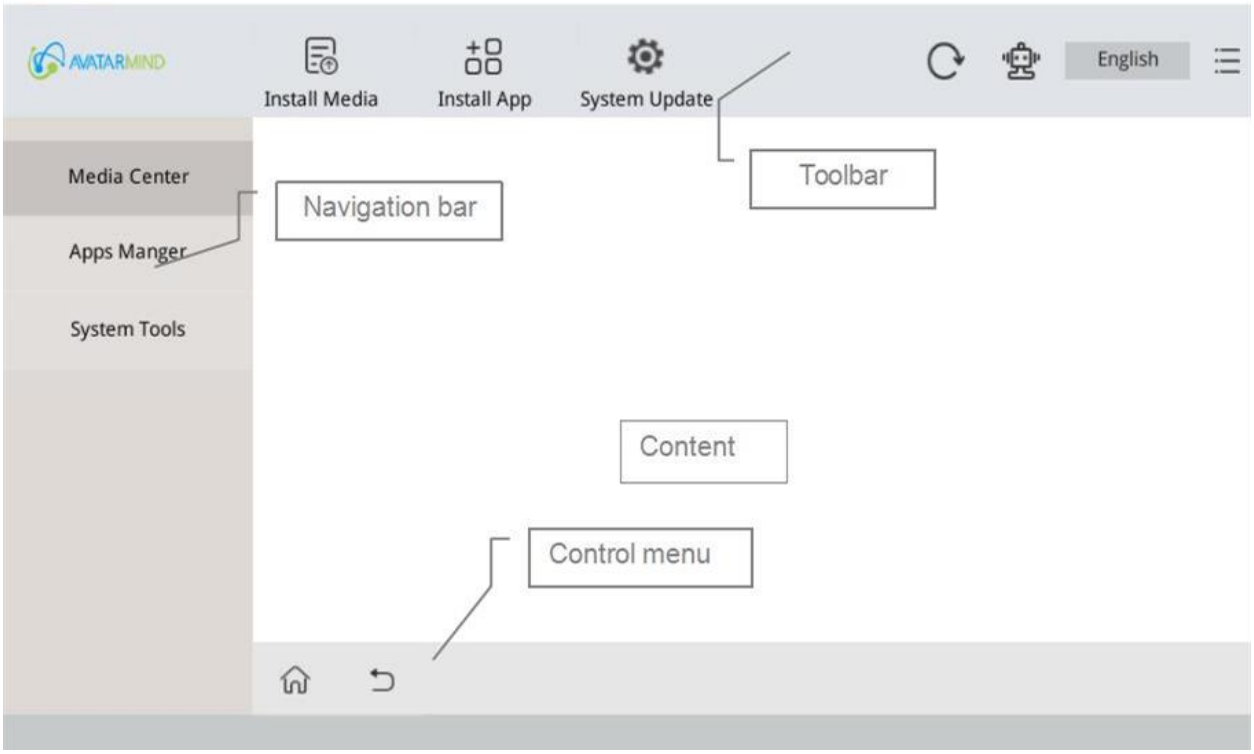


- **Exit**

  Exit feature is located at top right corner.
  Note: if iPalHelper is under upgrade, please DO NOT stop the process!!!

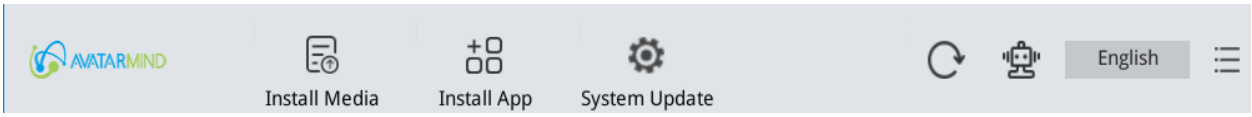
Overview

The home page of iPalHelper can be divided into 4 parts, as shown below.



- **Toolbar**

  The Toolbar is along the top of the iPalHelper app

  Install Media -- User can choose local media files on PC (*.arc, *.arm, audio, video, image) and upload to iPal internal storage.

  Install Apps -- APK files can be selected and installed on iPal.

  System Update -- User can upgrade or update iPal system by choosing iPal firmware (*.img) file or patch (*.ipp) file from local computer.

  Refresh Button -- Refreshs button is used to refresh the iPal device list and content area.

Shows the iPal device list.

Swtich the visibility of main menu.

- **Navigation bar**

  The navigation bar is on the left side of home page. There are 3 options to choose from:

  Media Center: display and manage all ARC files in iPal. Arc files are the content packages that can be displayed by iPal

  Apps Manager: display and manage all applications on iPal

  System Tools: display device info and system profile of iPal device. This is an advanced feature.

- **Content area**

  Displays the contents of the item selected in the navigation bar.

- **Control menu**

  The control menu is a strip along the bottom of the screen

Return to Home page of iPal system.

Return to the previous page or exit the running application on iPal.

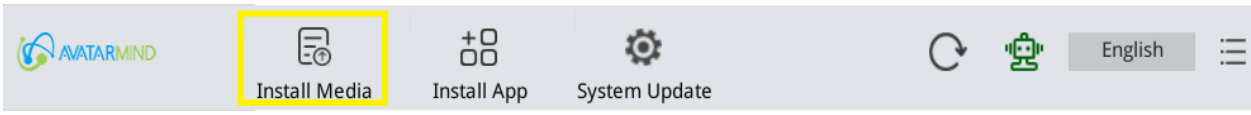
Run the selected application or media resource.

Delete/Uninstall the selected media or application.

Download the selected media resource to the PC.

Upload/Install Media on iPal

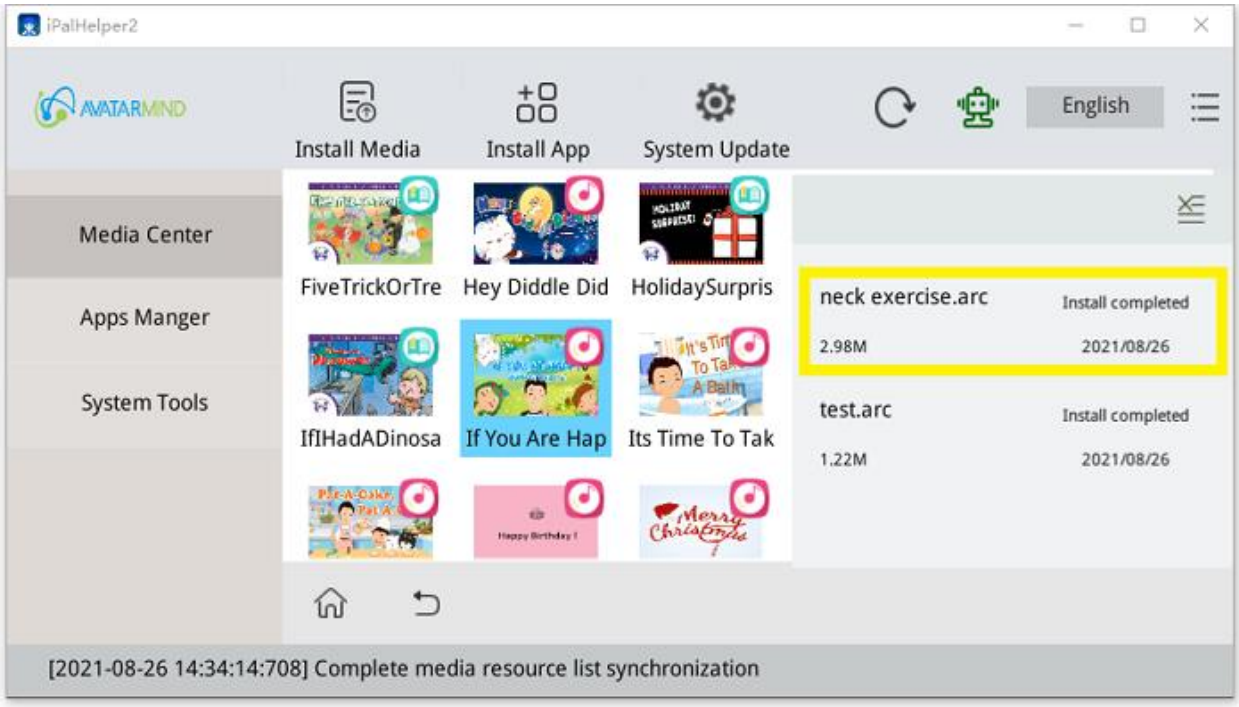
- **Select media file**



When this is selected a screen will pop up that will let you browse to the media you want to install on iPal.

- **Upload**

The uploading status will be displayed in the task list, as shown below.



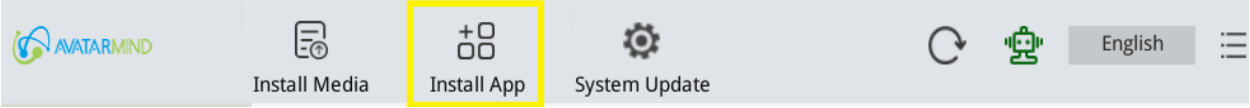
Note:

- Supports uploading Avatarmind custom content format ARC/ARM files, and uploading audio/ video/ image files, etc.
- DOES NOT support uploading zip package and directory.
- Filename or path can ONLY have characters in ASCII format, or the upload process will fail.

Note: the newly installed media may not show up on iPal until you restart iPal

Install Application

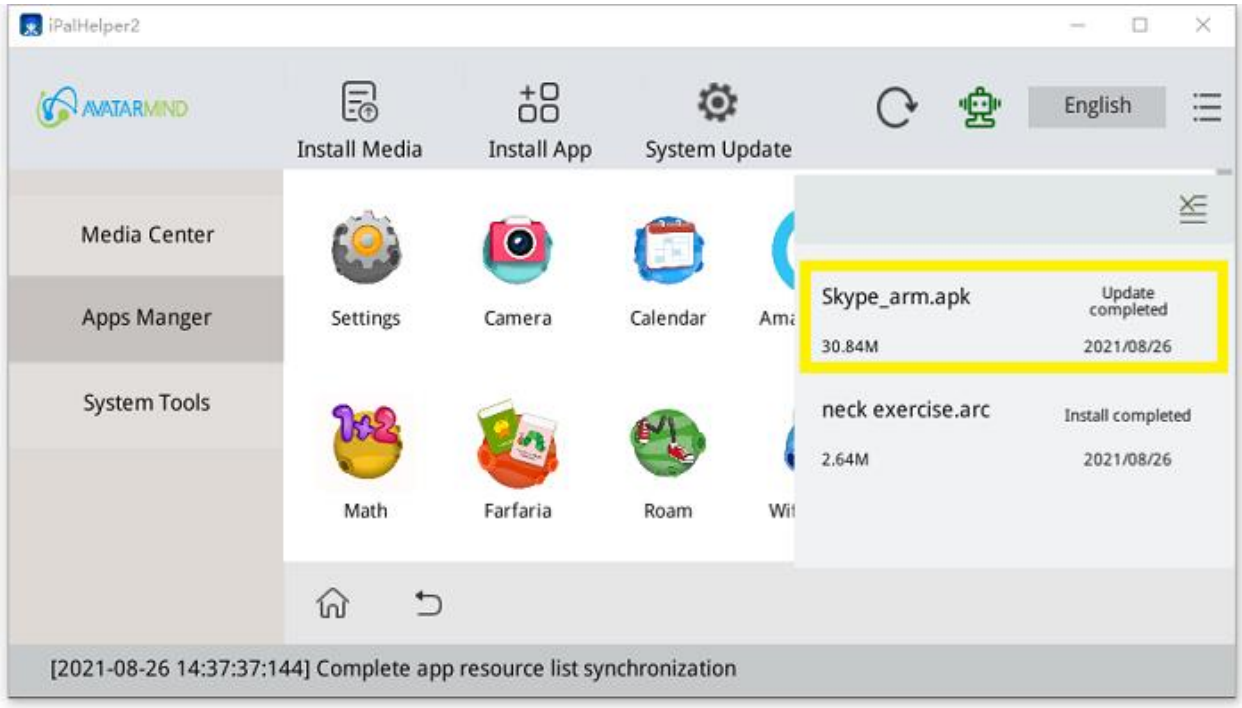
- **Select application**



When this is selected a screen will pop up that will let you browse to the application (.apk file) you want to install on iPal.

- **Install application**

  The status of the application will be displayed in the task list, as shown below.



Note: the newly installed app may not show up on iPal until you restart iPal

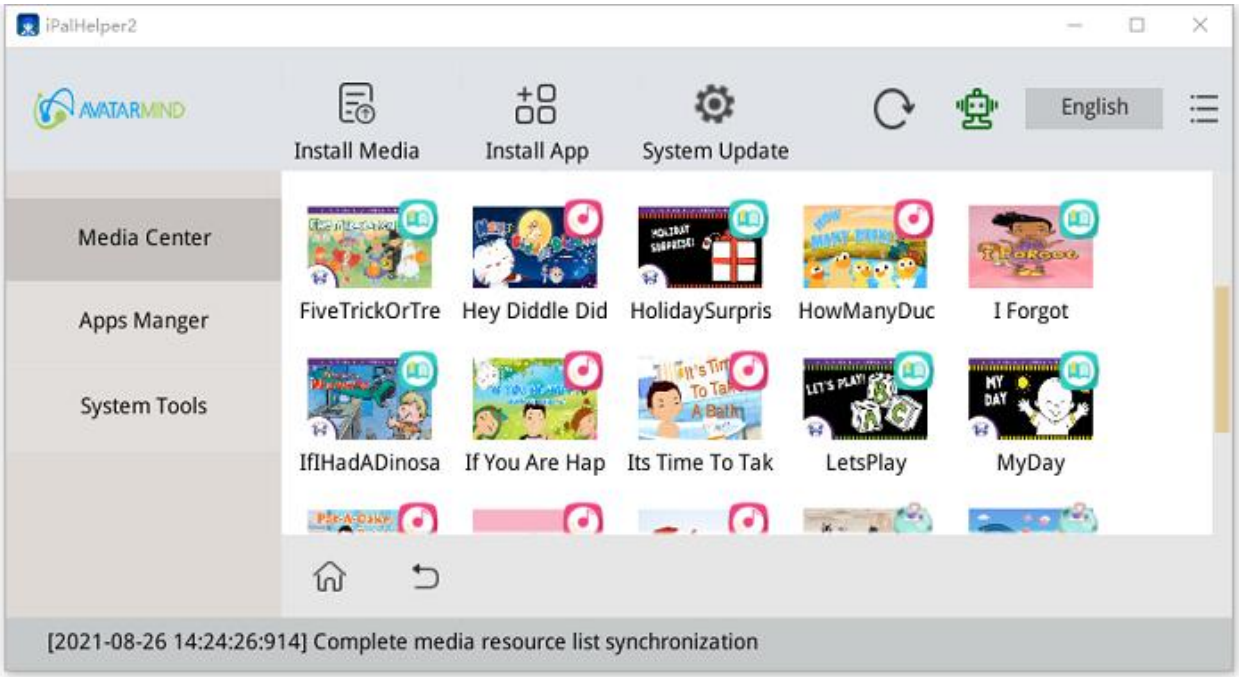
Media Center

Media List

As follows “Media Center” has been selected in the navigation frame at the left. The main screen then shows the media (songs, stories, etc.) on iPal.

You can delete an item using  at the bottom.

You can add a new media using 



Operations

Choose single media file and use the buttons below to control it.

 **Play**: play the selected media file on iPal.

 To stop the media playing click the return button on the left.

 **Delete**: delete the selected file from iPal.
(if this file is bundled from iPal system, iPal may need to be restarted for refreshing list)

 **Download**: download the selected file from iPal to the PC's local storage.
(**Note**: it cannot be downloaded to the root directory, such as D:\, which may cause the failure. Alternatively, the path can be D:\media\ as an example.)

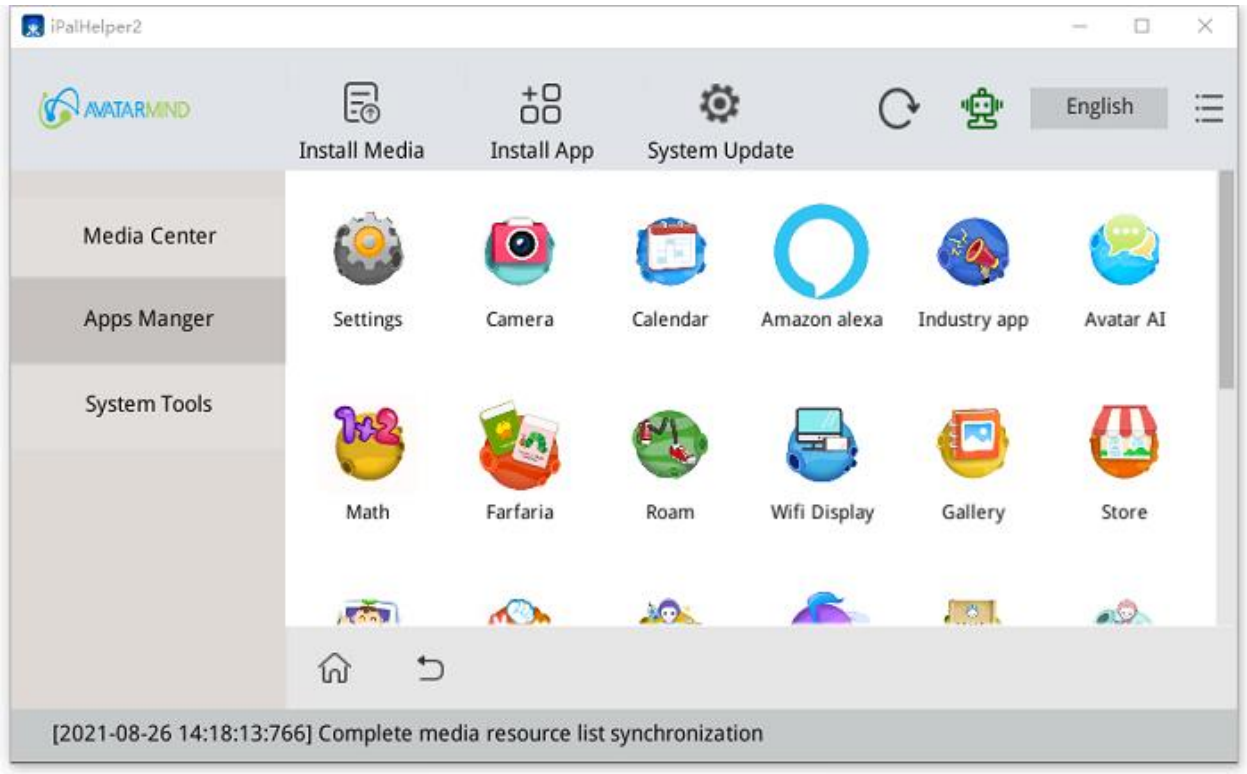
Application Management

Application List

As follows “Apps Manager” has been selected in the navigation frame at the left. The main screen then shows the applications on iPal.

You can a delete an item using  at the bottom.

You can add a new media using  in the toolbar along the top.



Operations

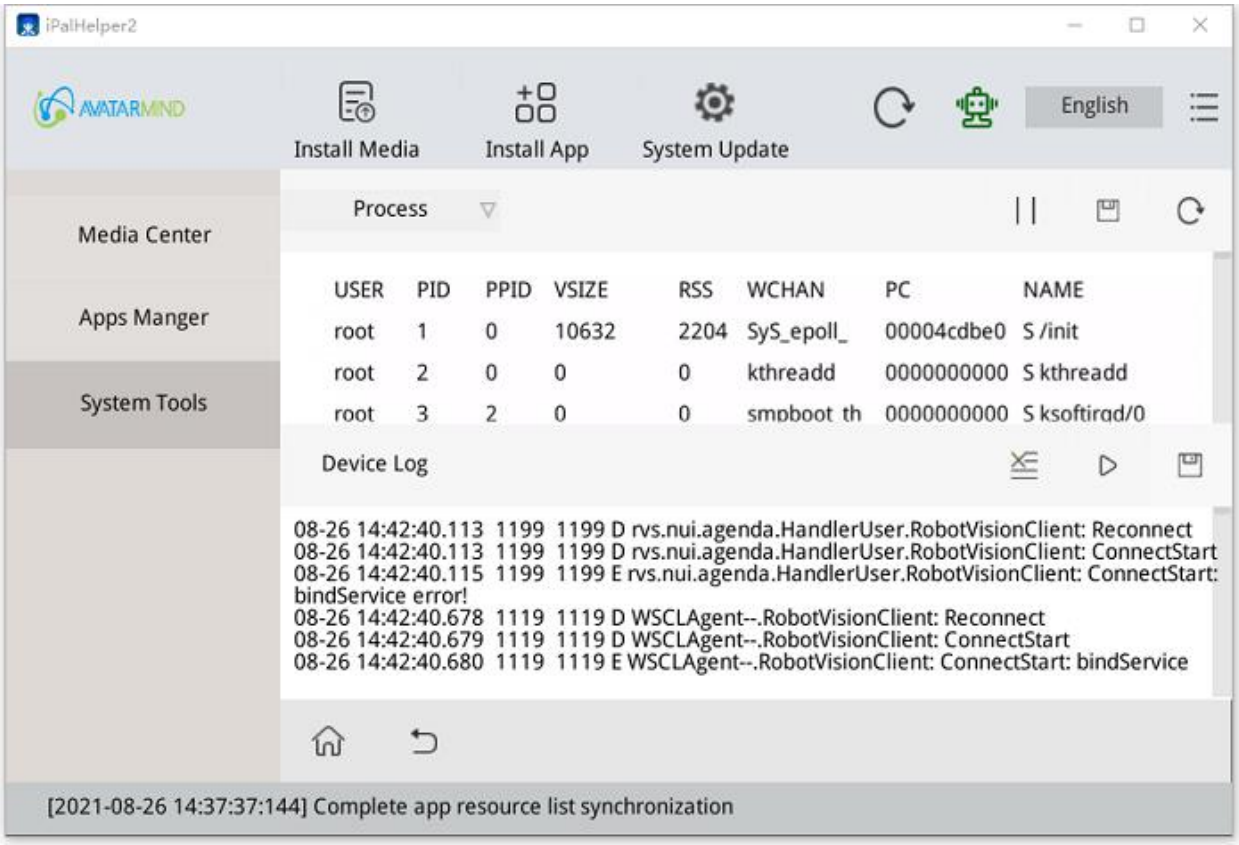
User can operate the selected application from above list by using the 2 buttons defined below.

 **Run**: run the selected application on iPal.

 To exit the application click on the return button on the left.

 **Uninstall**: to uninstall the selected application from iPal.

System Tools (for advanced use)



System Log

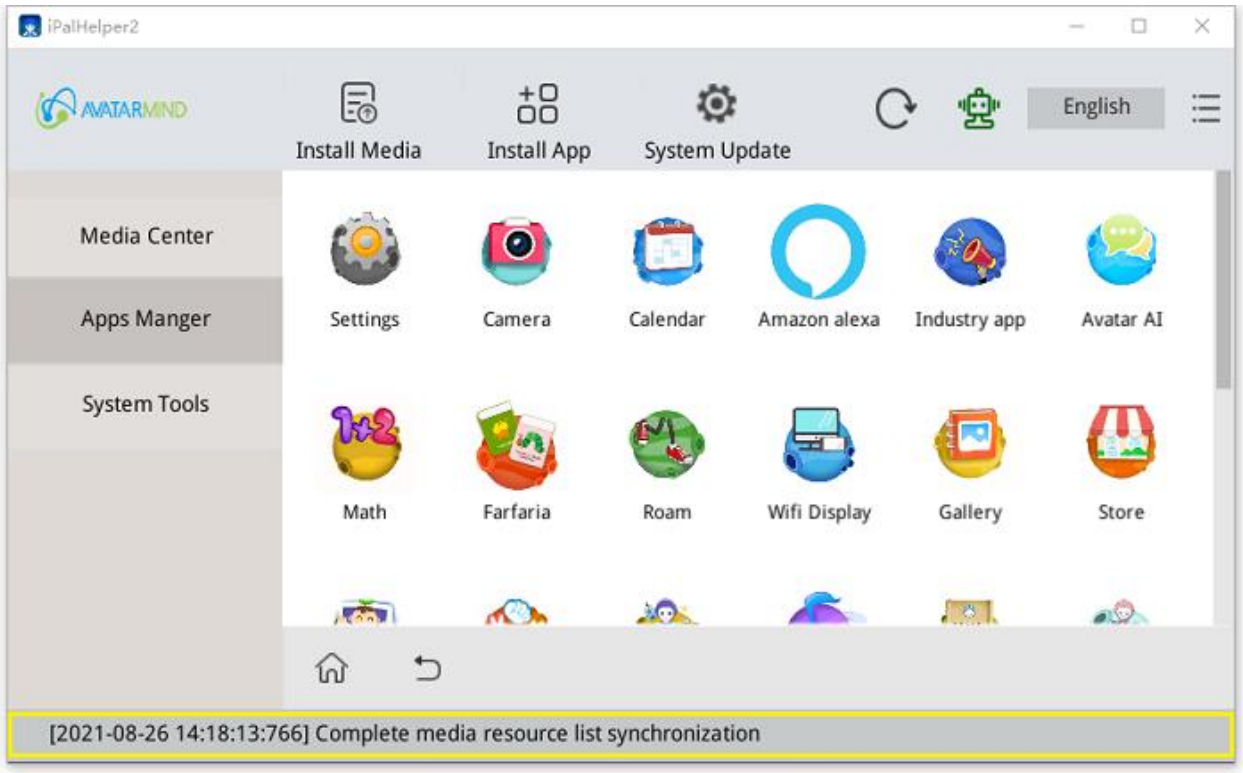
To show the system log select

To start capturing the log select at the top right

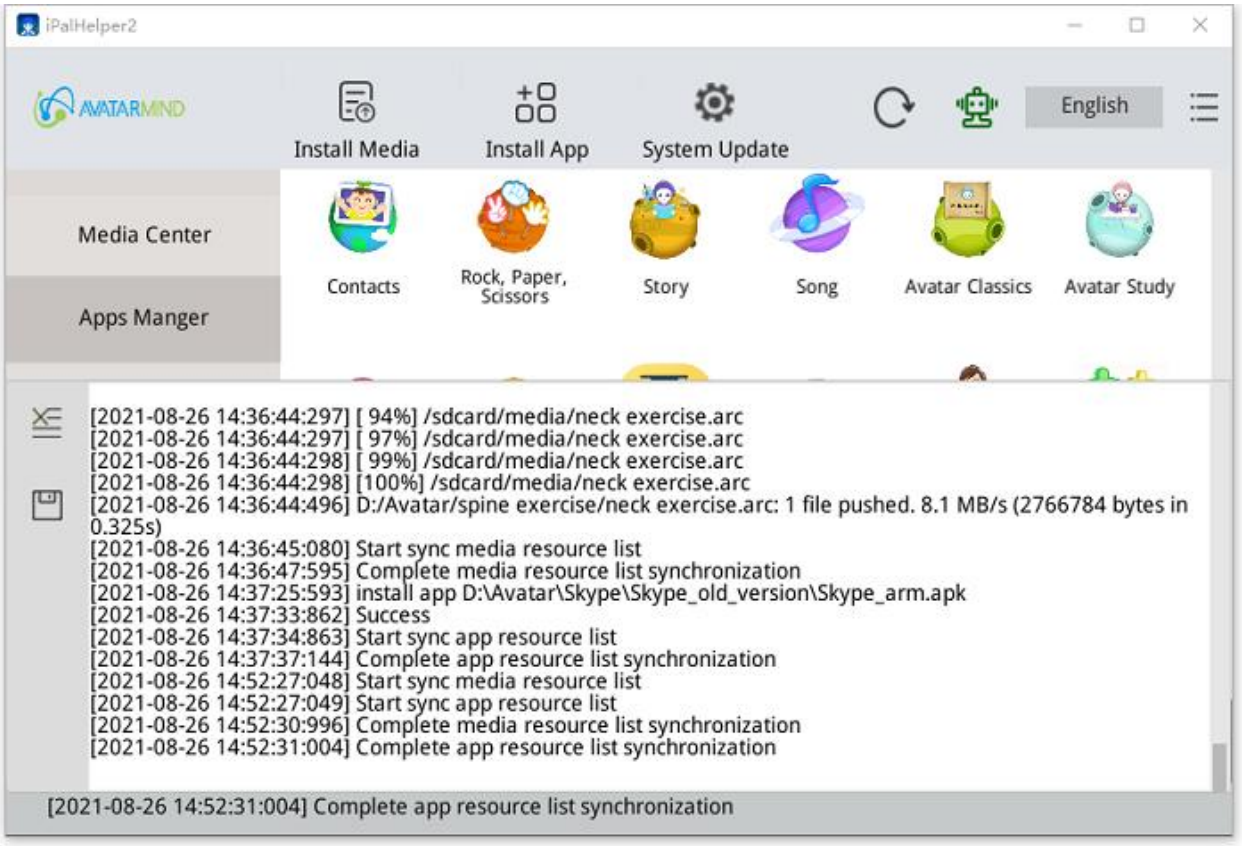
To stop capturing the log select at the top right

To save the log to a local file on the PC select

Installation Log



To show the app/media installation log double click on the dark horizontal band at the bottom of the screen. This will bring up a view like that shown below. The log for the item just installed will be shown at the bottom.

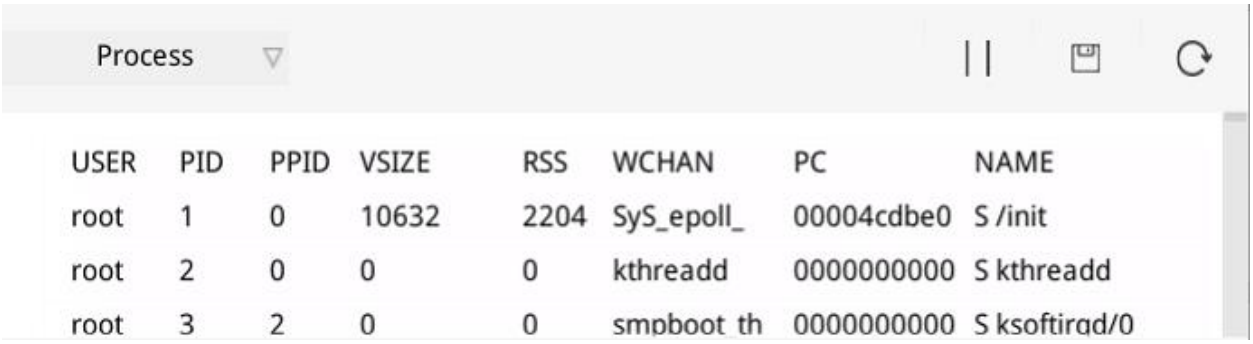


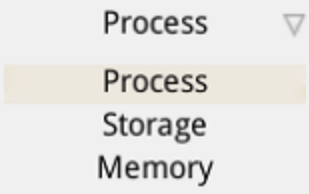
Along the left hand side of the bottom screen you see 

To show the system log select 

To save the log to a local file on the PC select 

Tools





User can check the process status, memory status, storage capacity, file system info and etc., via the feature list above.

Stop the process (Only available in the process list page)

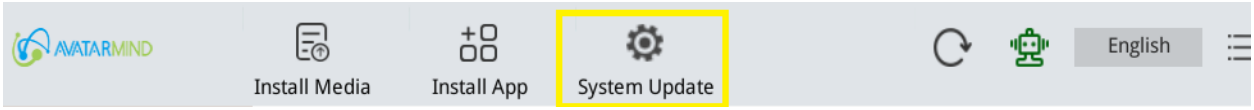
To save this information to a local file on the PC select

To Refresh the list

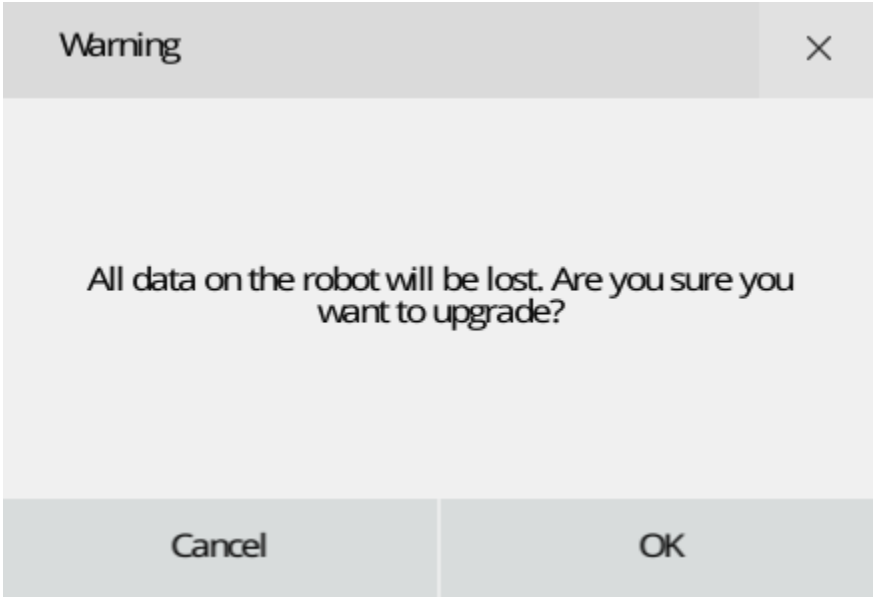
System Update

Updating the System Firmware

- **Select the system update button and then browse to find image file(*.img)**



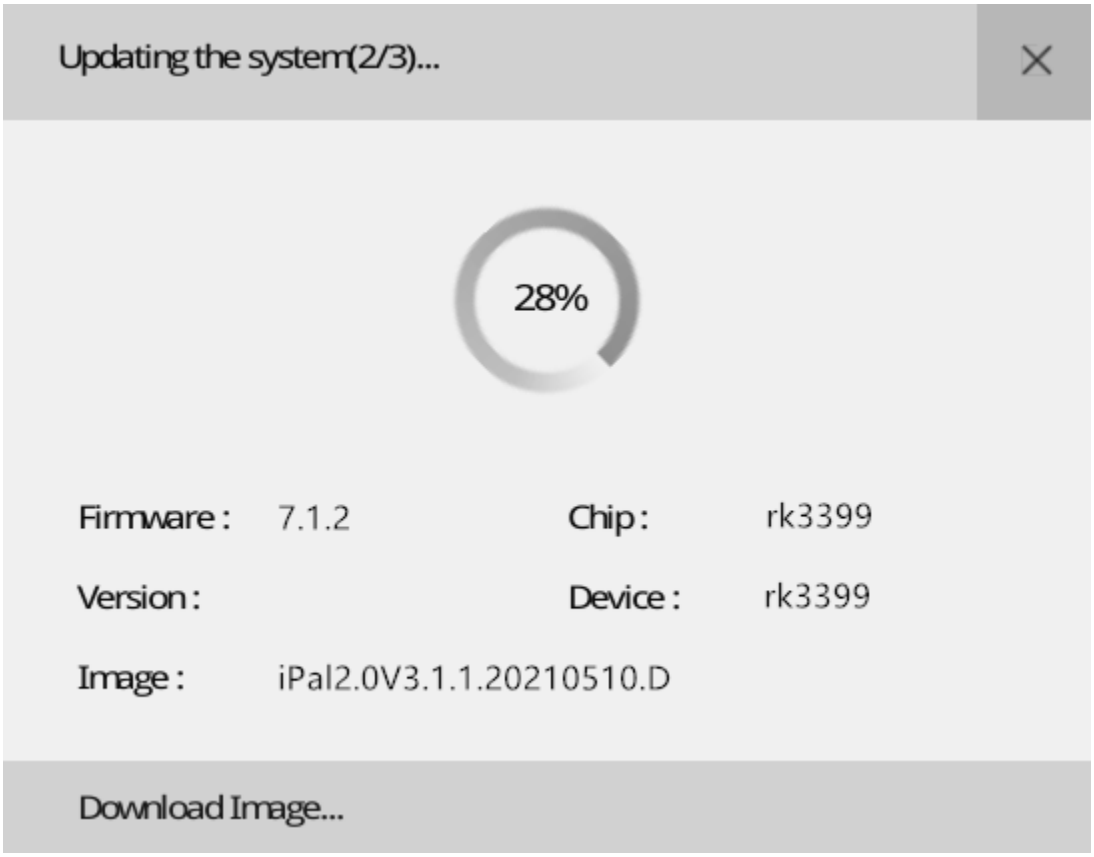
- **Confirm that you want to do the upgrade**



Note: Please save all data before upgrading the system. If not, all user data will be lost. Do not shutdown iPal while the upgrade is in progress, this will result in an installation failure.

- **Start the system upgrade**

Select OK in the above screen and the following screen will appear.



Once completing the upgrade, the robot will be upgraded and automatically reboot. Then follow the setup wizard on the screen of iPal.

More Functions

Shell script

Create shell script

Any tool with verbal edit function can create/edit the shell script. Only 2 types of grammar are supported.

- Variable assignment

  Language:

  <varname> = *<varvalue>*|*<@varname>*|*<OpenDialog()>*|*<SaveDialog()>*
  OpenDialog() gets the open local file name
  SaveDialog() gets the file name saved to the local

- Variable reference

  @varname

Example:
Adb push @varname /sdcard/

- ADB command

  Language:
  adb <*arg1*> … <*argn*>;
  Note: it does not support output redirection (>>) operations.

  Example:
  adb shell ls /sdcard/

  Examples:

  Eg 1:
  ;; sample1.ish
  ;; This is sample shell
  ;; Comment
  FileName=SaveDialog()
  LocalFile=@FileName
  adb pull /sdcard/media/xxx.arc @LocalFile

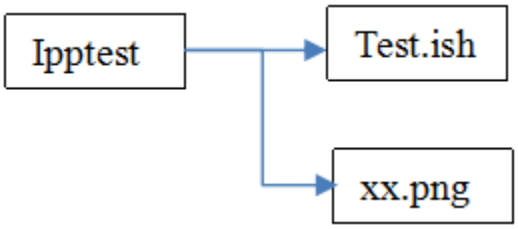
  Eg 2:
  ;; sample2.ish
  ;; This is sample shell
  ;;
  adb remount;
  SOFILE=OpenDialog();
  adb push @SOFILE /system/lib/
  adb reboot

Update package

Put the script and the required resource files together in a zip package, and update the system through iPalHelper.
Note: The suffix name of the zip format file needs to be changed to ipp, otherwise iPalHelper does not support it.

Example:

The directory structure is as follows:



Test.ish：

;; this is test shell

adb remount;
adb push 2.png /sdcard/media/

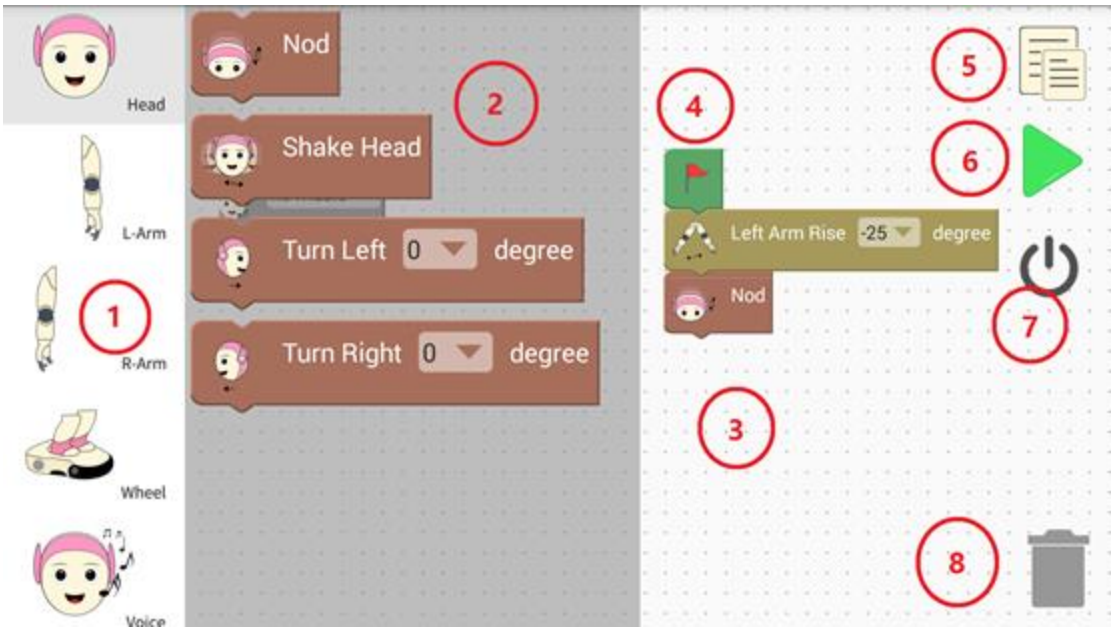
Overview

The purpose of iPalProgrammer is to educate children on the basic concepts of programming and logic. iPalProgrammer is developed by AvatarMind using iPal as its platform and provides children with a drag-and-drop interface. Programs created in iPalProgrammer are converted to RobotScript ('.rs') files. iPal will parse the '.rs' file and send commands to its motors accordingly. RobotScript is a scripting language similar to JavaScript. iPalProgrammer provides the ability to save and upload programs. Unfinished programs can be saved, and existing programs can be loaded from robot. Programs can also be uploaded to the iPal store. Uploaded files will be assigned a link for other users to download.
iPalProgrammer supports PC (Windows; Linux; Mac) as well as Tablet (Android). Please download and unzip the correct version for your OS: **Windows** | **Linux** | **Mac** | **Android**

Module

Home page and functionality modules of iPalProgrammer is shown as below:

1. **Categories**: The left column lists movable body parts and other categories
2. **Actions**: Each category has a list of actions it can perform. They are listed here
3. **Workspace**: The area where blocks are located. Drag blocks into the workspace and connect them to create a valid program
4. **Start**: Only blocks connected to the start flag are valid
5. **Menu**: Menu options
6. **Run**: Run the current program in the workspace
7. **Exit**: Exit iPalProgrammer
8. **Trash**: Delete blocks by dragging them to the trash icon



Module Details

Workspace

The workspace is the main coding space. It contains the start icon and space to develop your program. Valid action blocks that will run are colored, while invalid blocks will be grayed-out. Blocks are run sequentially starting from the start icon, and each block is run individually, only starting after the previous one has finished. Select a body part from the menu on the left to open a selection of movement action blocks for that body part. Drag your desired block into the workspace and connect it to another block. Valid connections will snap to each other. Note: Some blocks allow you to input angles. Once in the workspace, tap the number to change it to a desired angle. You can set some angles to be negative. Additionally, input angles are not additive. For example, if you raise the left arm 30 degrees, the arm will raise 30 degrees. If you add another block to raise the arm 30 degrees again, the arm will remain raised at 30 degrees.

Voice

Select Voice in the left-hand column. Select one of the options below. Music: iPal can play stored mp3 music files. Music files should be stored in '/sdcard/Music'. Text-to-Speech: Once in the workspace, you can edit the text in this block. When the block is run, iPal will read out the text.

Expression

Select the Expressions menu. From here you can alter iPal’s expression to simulate a variety of different moods.

Combination

iPalProgrammer pre installs some motion combinations. Motion combination is defined as a series of motion with each has specific meaning.

Control Logic

Select the Control menu at the bottom of the left-hand column • “if/do”: Insert a condition on the right side. If the condition is met, run the contained action block. If the condition is not met, move on to the next block • “if/do/else”: Similar to “if/do” blocks, but if the condition is not met, run the block inside the “else” section • “Circulation x times”: Run the contained action block x times. You can set x to be a desired number of repetitions • “Repeat/do”: Insert a condition on the right side. You may select to repeat “while” or “until” the condition is met • “Meanwhile”: See below • “Reset”: Reset all of iPal’s body parts to the default state • “Sleep”: iPal will remain in its position for x seconds before moving to the next block. You can set x to be a desired number of seconds

Meahwhile: iPalProgrammer provides the capability to combine actions, for example moving both arms at the same time. Select the Control menu on the left-hand side by scrolling down. Drag a meanwhile block into the workspace. You can now drag multiple action blocks into the meanwhile block to have them run simultaneously. Note: Some actions cannot be performed simultaneously. For example, you cannot raise and lower an arm at the same time. If two conflicting actions are inserted in a meanwhile block, the two actions will be executed sequentially.

Run Button

Tap the Run icon to run your program. While the program is running, a pop up will appear on the screen.

Exit Button

Tap the Exit button to exit iPalProgrammer.

Menu

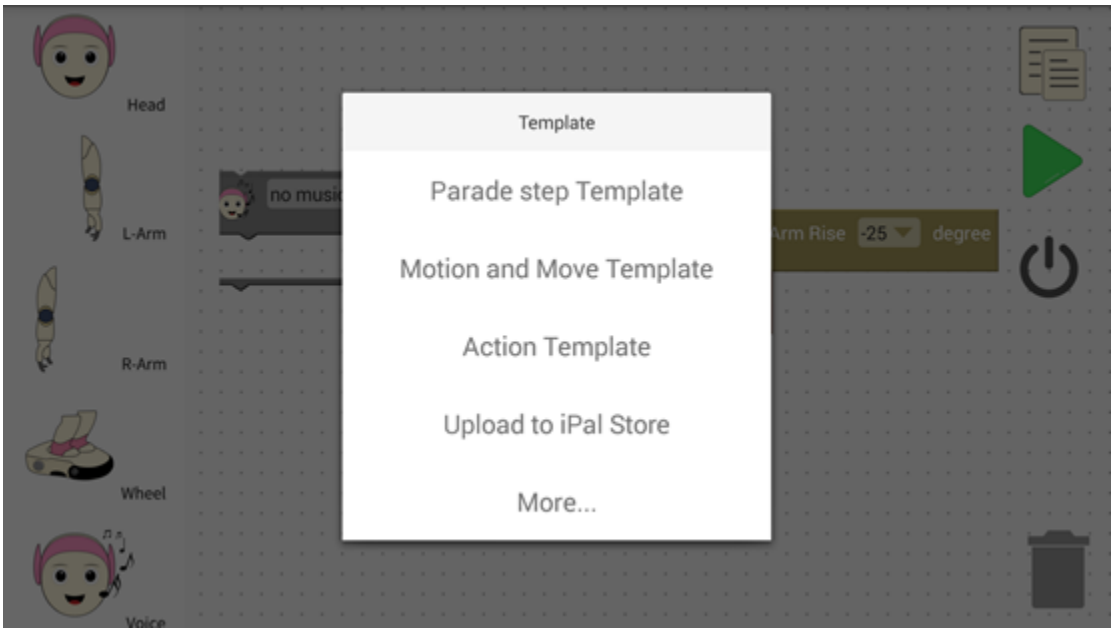
Template

iPalProgrammer provides a variety of preloaded template (example) programs for you to use. Tap the template icon to open the templates menu. Tap on a template to load it into your workspace.

Note: Loading templates will clear the workspace, so make sure to save your work beforehand.

You can upload your own programs to iPal store. Tap the Template icon and scroll to the bottom of the template menu. Tap “Upload to iPal Store”. Your program will be reviewed by an administrator. Once approved, your program will be available for others to download as a custom template, as

shown below:



Save Template

To save your program locally, touch and hold the Template icon (at the top right). A popup will appear, prompting you to enter a file name. Enter a name for your code, then tap “Save” to save your program. You will see your program when you select “more” in the figure above. To load a saved template, tap the Template icon and scroll down to “More…”. Tap “More…” and select the program you would like to load into your workspace.

Trash

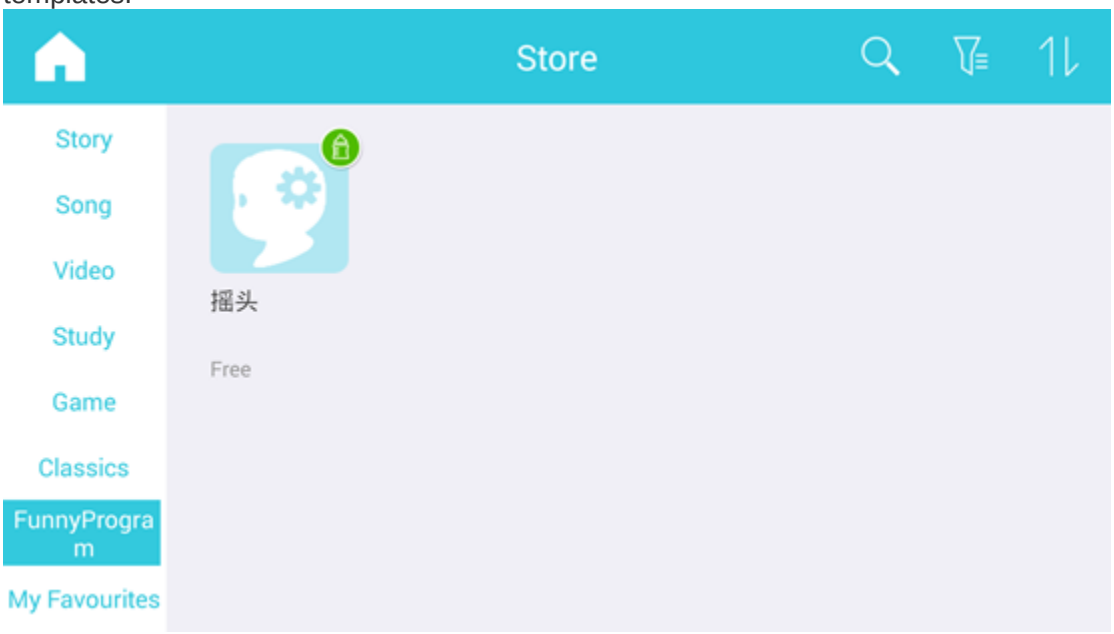
iPalProgrammer provides two methods to delete work:

- **Single Block** To delete a single block in iPalProgrammer, drag the block to the Trash icon. When the Trash icon changes to an open trash can, release the block to delete it.
- **Clear Workspace** To clear the whole workspace, tap and hold the Trash icon. A popup will ask if you would like to delete all, tap “Delete” to delete everything in the workspace. Note: The Start icon cannot be deleted.

Download Template

Download templates from **Template** in the iPal store. The templates are downloaded to '/sdcard/.funny_program'.
In iPalProgrammer, downloaded templates are accessed in the same fashion as the preloaded

templates.



Overview

This document will explain how to use a USB connection to update a robot's firmware version. Updating the robot firmware needs two tools, a firmware image and the update tool. The update steps are described below.

Steps

Preparation

- update.img: an AvatarMind robot firmware update image, used to update a robot's firmware (system) version. The current Android version is Android 7.1 and firmware is v3.4.3, please download the latest firmware **here**.

Installation

Please Download the BurningTool installation file with AvatarMind provided **link**. Unzip the package to your local computer. Must be a Windows platform. You will see the following file:

- BurningTool_setup_v4.0.2.exe, installation file

Click the installation file BurningTool_setup_v4.0.2.exe and you will see the Figure below:

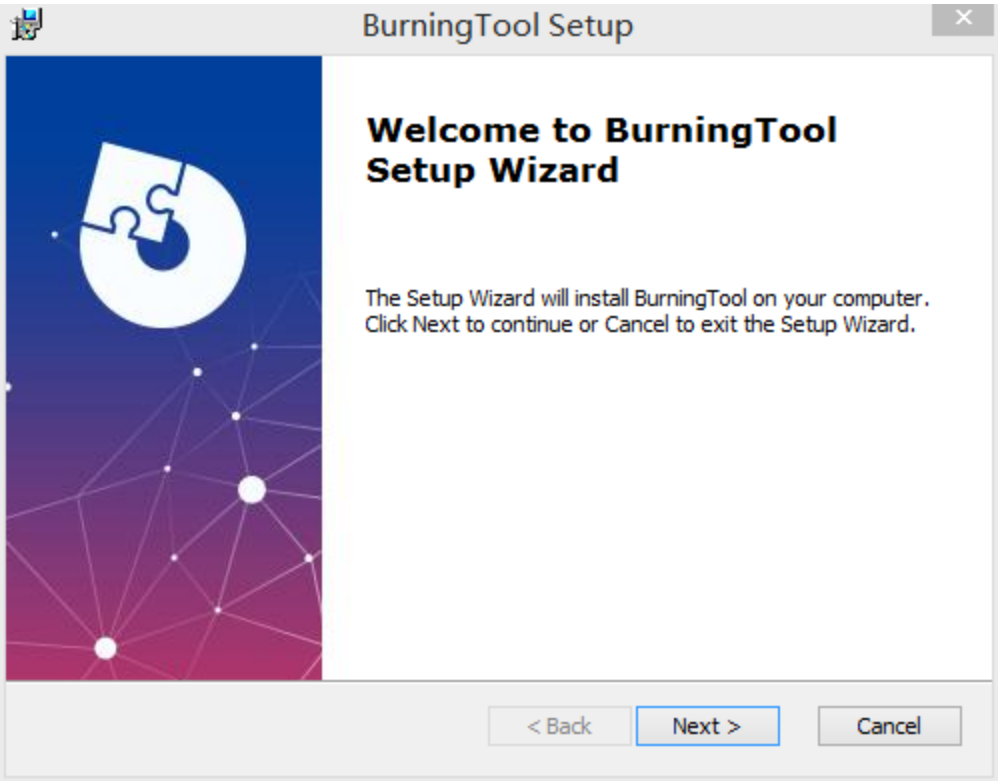


Figure 1.1: BurningTool Setup Wizard

Follow the Setup Wizard to complete the installation.

Login

Double click the icon of BurningTool in your installation path. Type in as the Account. Click "Login" button to enter the home page of the BurningTool.

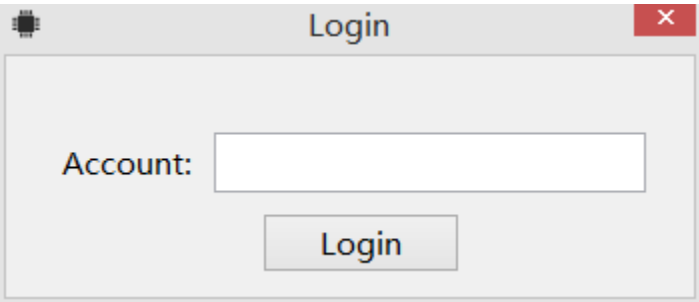


Figure 2.1: Login

The first time using the BurningTool the user has to type in the login code. After that, the code will be saved automatically.

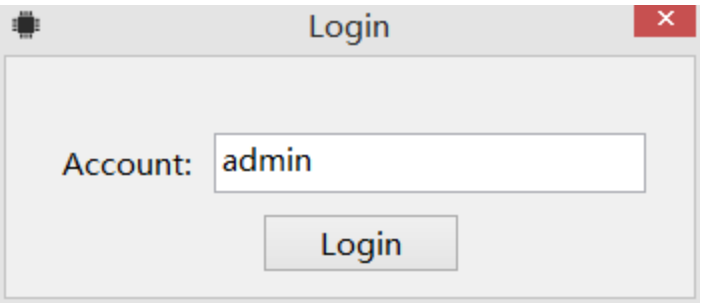


Figure 2.2: Login Code Saved

Upgrade iPal Firmware

Home Page

After login, you will see the home page of BurningTool. Please select  **at the top left as shown below.**

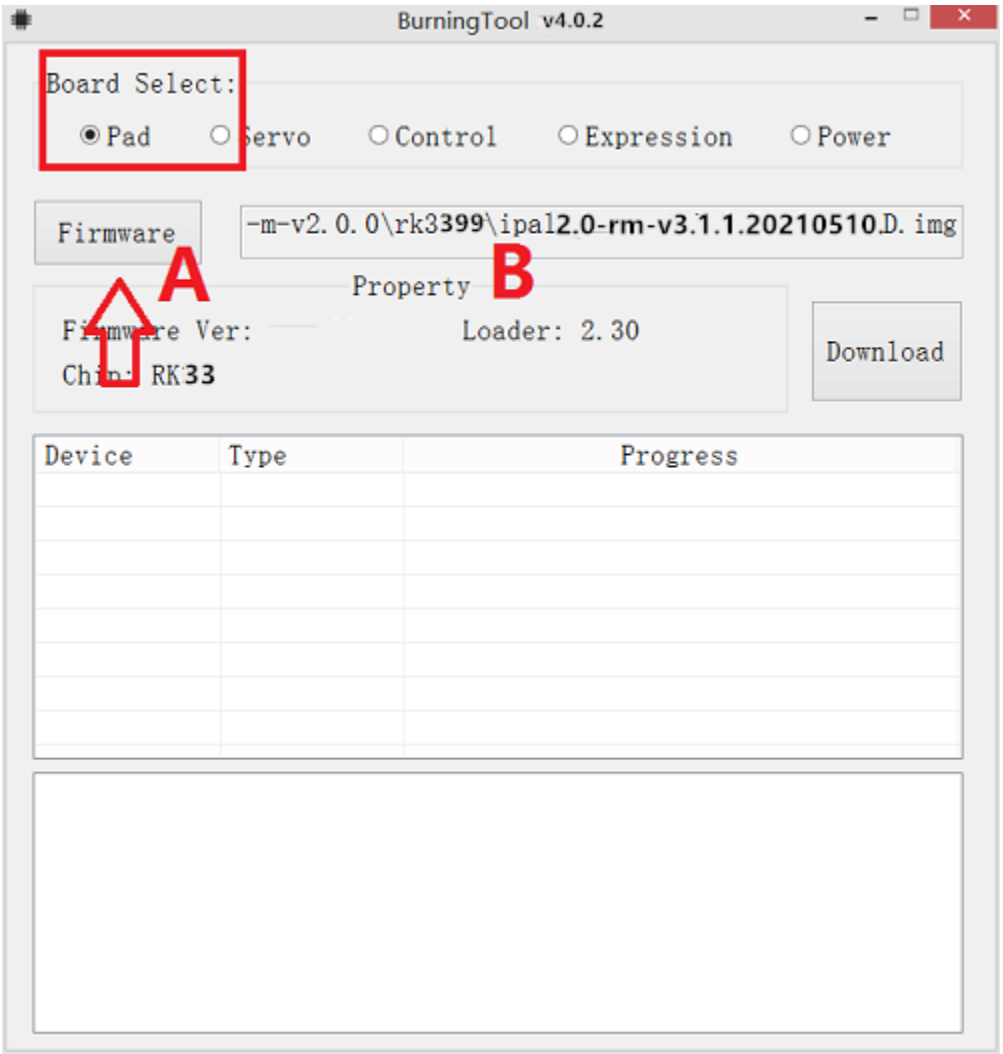
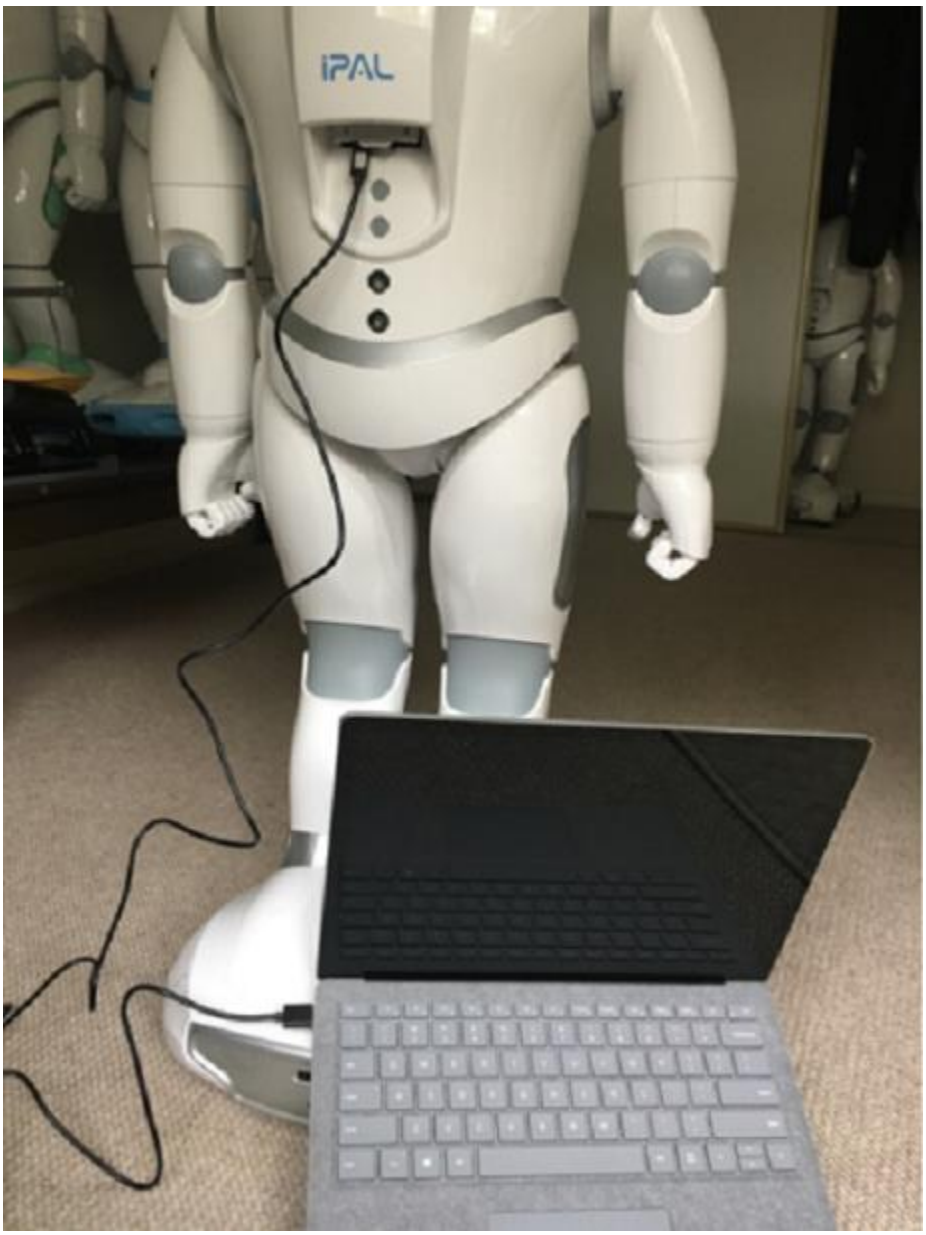


Figure 3.1: BurningTool Home Page

- Click the 

button to browse your computer's file system to find the iPal robot system image firmware file. See A in above picture.
- The Filename must contain string "rm" or BurningTool will not recognize the file. Firmware names released by AvatarMind should already include this.
- Once you select the firmware by browsing you will see the file path appear in the window in above picture – see label B.

- Please connect the iPal robot by a USB type-C cable to your PC. See the picture below.

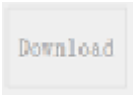


To connect to iPal open the flap at the back.

The figure below shows the connector on iPal outlined in red.

- When iPal robot is connected to PC, BurningTool will detect it and show on the screen under “Progress” in the bottom window, as shown in the figure below.

Loading

When user selects the firmware, BurningTool will load the firmware file. This step may take several minutes. Please wait until it finishes.The 

button will darken slightly when complete.

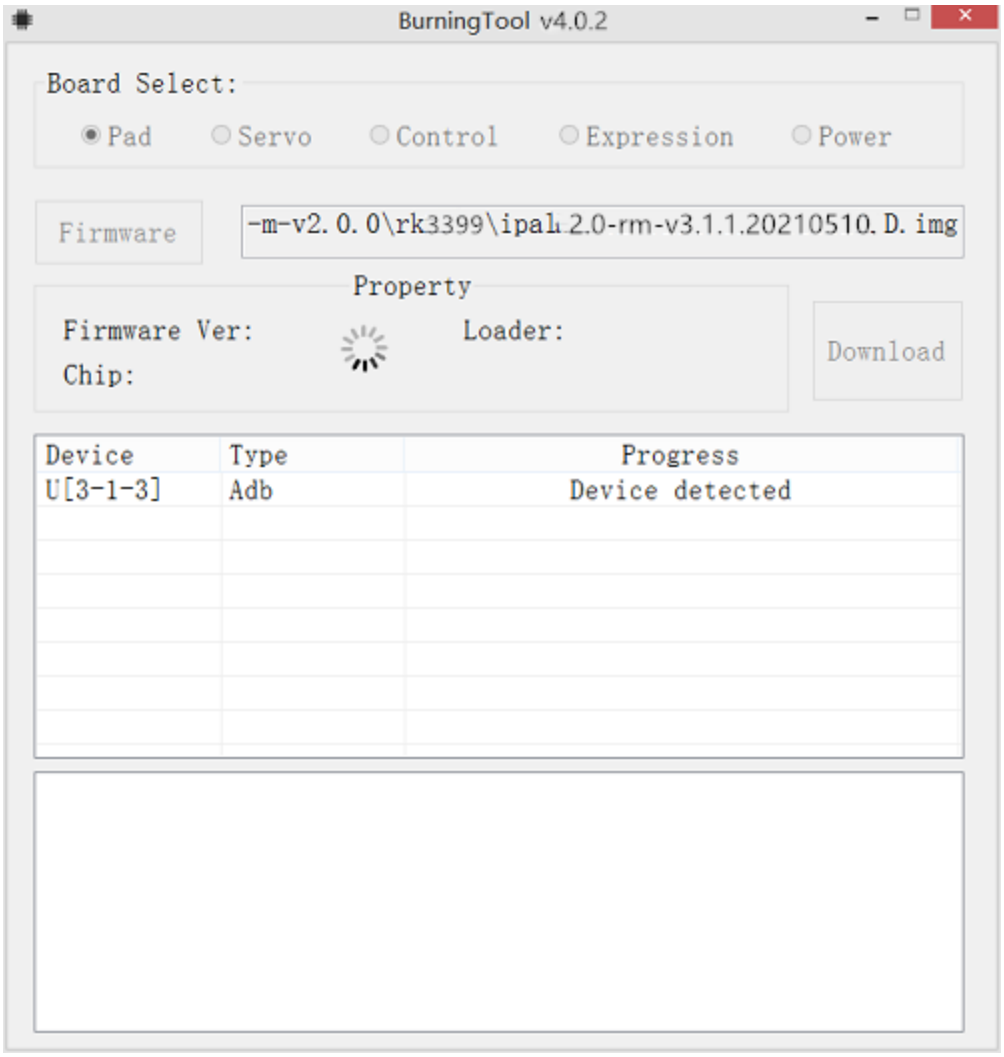
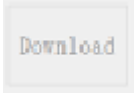


Figure 3.2: Loading Firmware

Burning

- After BurningTool loads the firmware file, click the 

button on the screen. Burning Tool will start the firmware installation and user can see the loading and verification process progress under “Progress” in the bottom window in Figure 3.3. When completed iPal will automatically reboot.

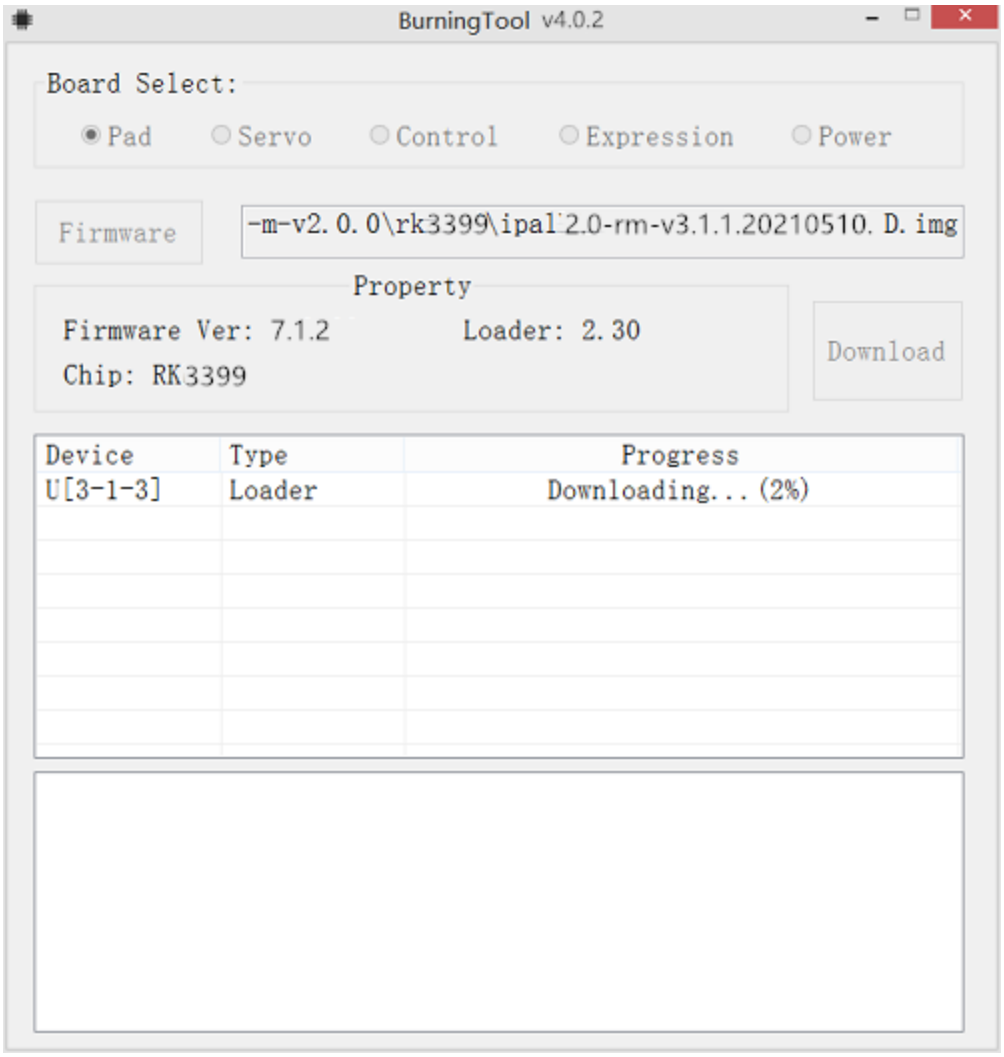


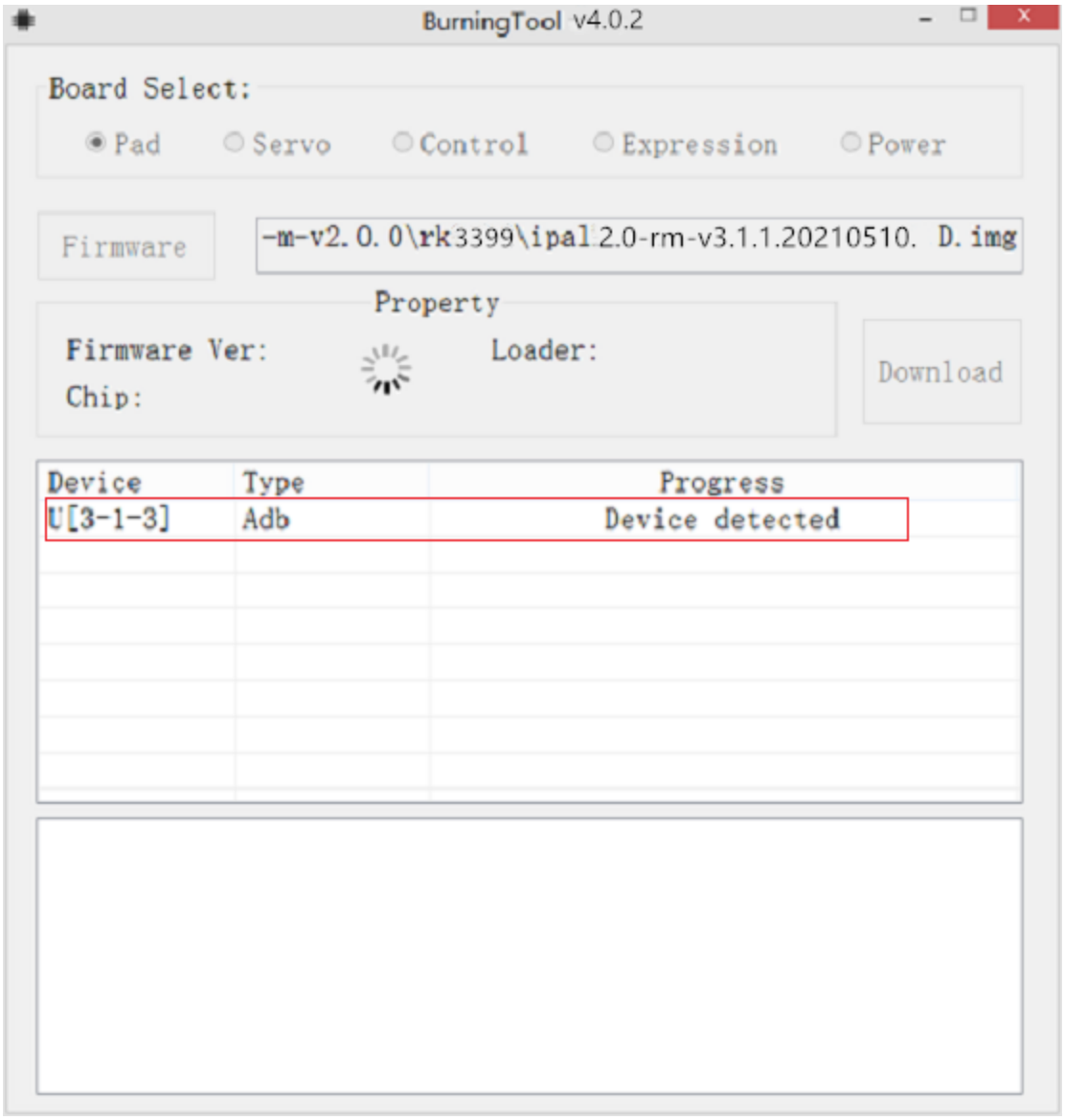
Figure 3.3: Upgrade

Note: When user clicks the “Download” button, the screen on the iPal’s chest turns dark but the facial expression remains on. Please see Figure 3.4. This condition means iPal robot has been set to the reboot mode. **DO NOT** turn off iPal robot while upgrading. iPal robot will be restarted automatically when upgrade is completed.

Figure 3.4: iPal Status while Upgrading

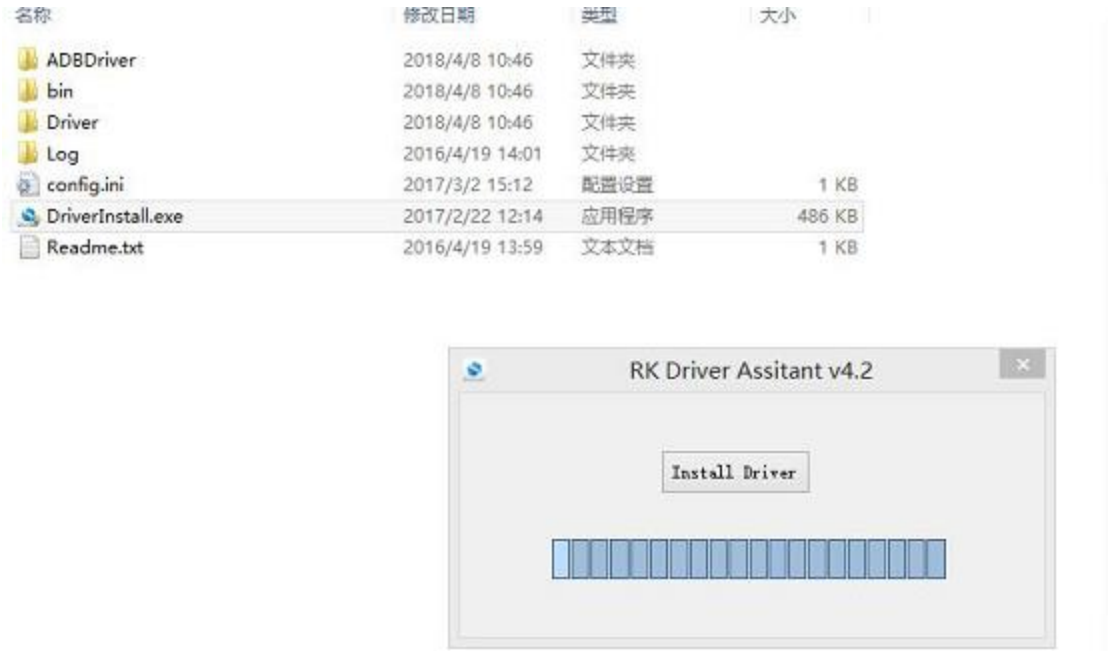
Upgrade problems

- Can not find ADB device

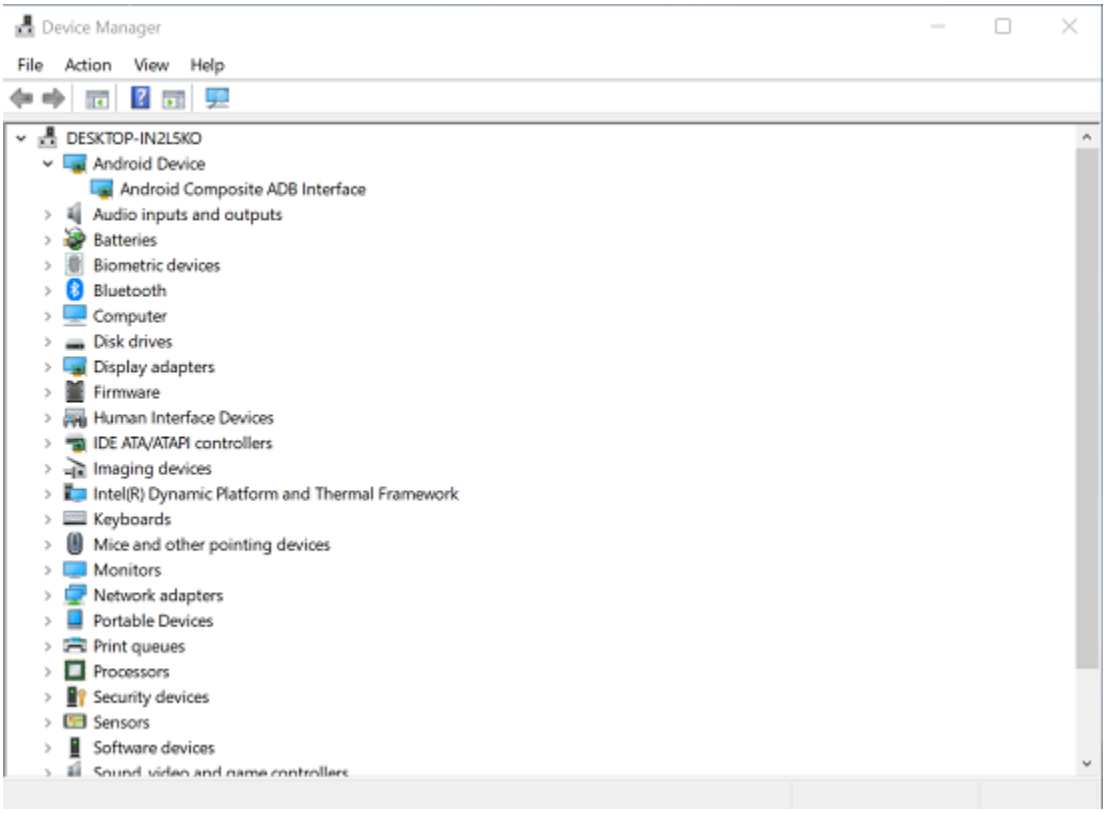


If the red box in the above figure is not displayed, the tool is not compatible with the PC and the ADB driver is not installed. You can try to change the PC or install the driver manually.

Method 1, run the driver installer **RK Driver Assistant V4.5**.

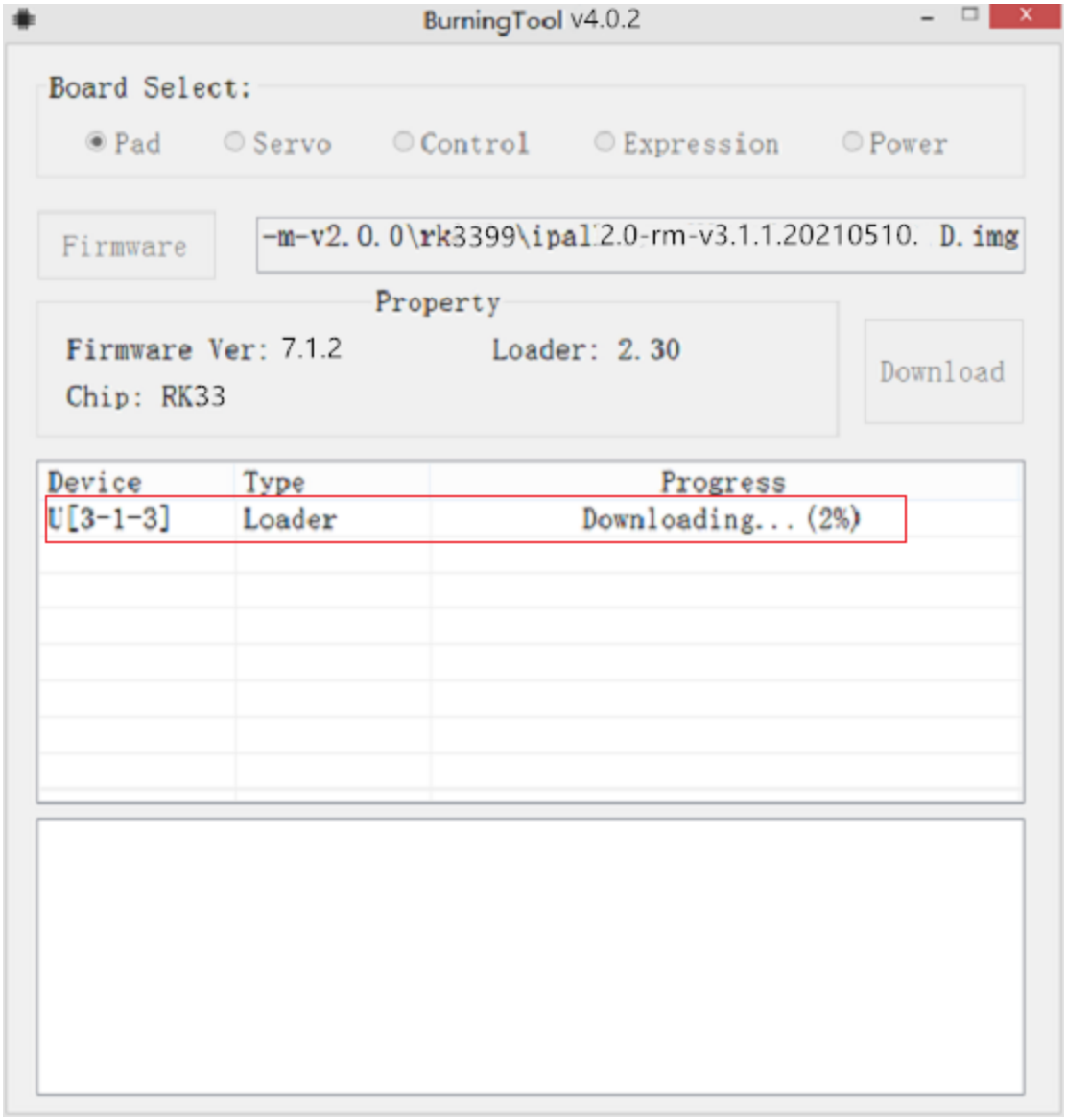


Click Install Driver. After the installation is successful, plug in the USB cable connected to the PC and open the device manager. You should see the Android Device text. See the figure below.



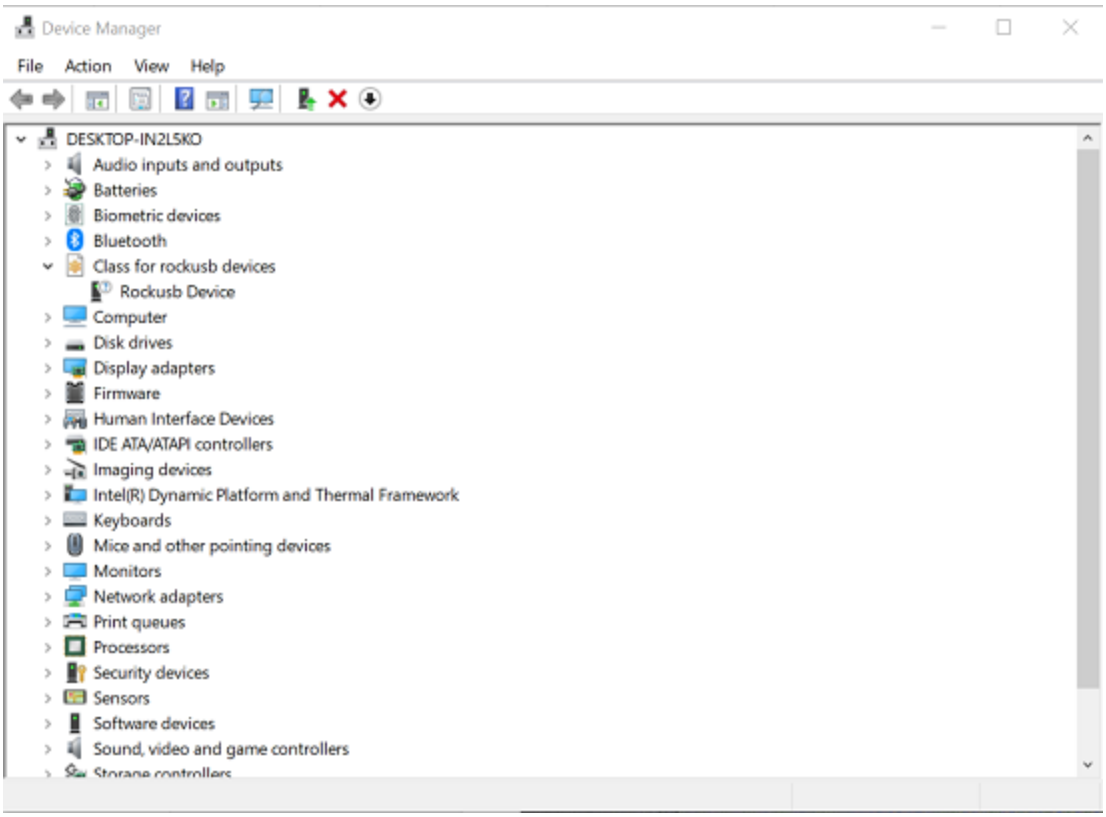
Method 2, run a third-party Android assistant, such as **91 assistant**, you can copy and paste the following link to the browser to download
it: `http://bos.pgzs.com/wscdn/assistant/wyx/pc/91assistant_pc_v6_1_2019081910577.exe`

- Can not find Loader device

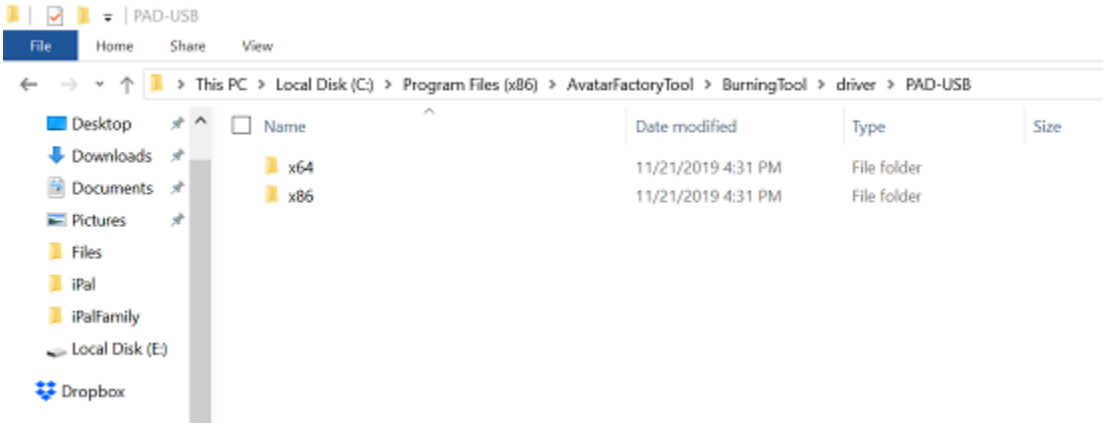


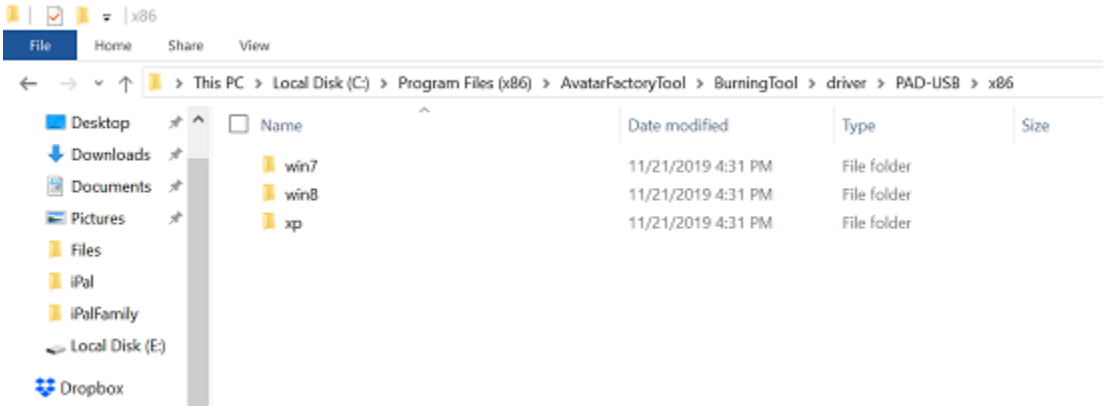
After clicking Download, the content shown in the red box above does not appear, indicating that the tool is not compatible with the PC, and the Loader driver is not installed. You can try to replace the PC or install the driver manually. The CPU requirements of the PC are Intel, such as X86, X64.

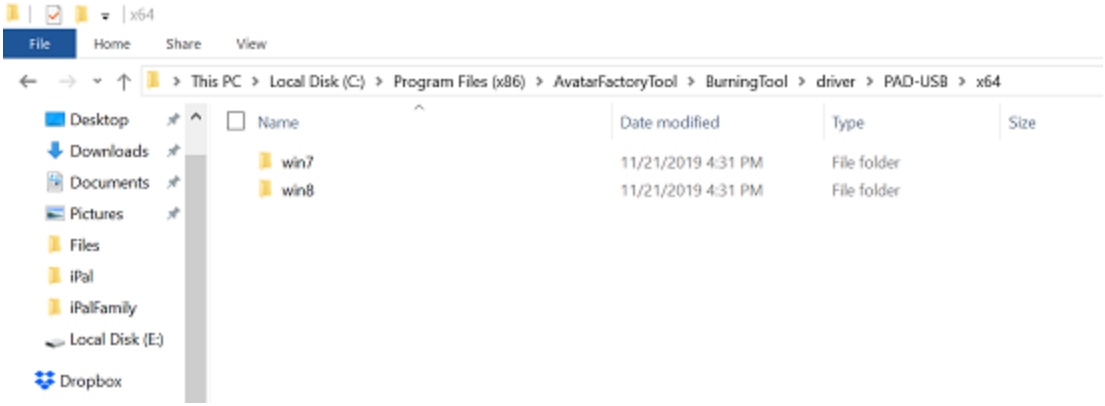
Manual installation via Device Manager, as described below.



The path of the driver is as follows, the CPU type of the PC is selected. If it is Intel 64-bit, select X64; if it is Intel 32-bit, select X86; if the system is Windows 7, choose win7, if it is Windows 8, choose win8, if Is the choice of Windows10 Win8.







Completion

When the download is complete, iPal will automatically boot up. You will then see a series of setup screens for the initial configuration of iPal. Please see the Quickstart manual for information on the setup process.

How to install the Loader driver manually?

- Prerequisites: The CPU requirements of the PC are Intel, such as X86, X64.
- Use the following steps:
  1.Click BurningToolV4.0.2 **Download** or input **“adb reboot bootloader”** in Dos shell window. The robot will show **black screen**.

  2.Open the **Start menu** and search for **device manager**.
  3.Select the top result.

  4.When **Device Manager** opens, right click Machine and click **Scan for hardware changes**.

  5.Expand the branch for the other devices you want to install.
  6.Right-click the device and select **Update driver** from the menu.

  7.On the following screen, select the **Browse my computer for drivers** option.

8.Click the **Browse** button and navigate to the location of the driver installation files. The path of the driver is as follows, the CPU type of the PC is selected. If it is Intel **64-bit**, select **X64**; if it is Intel **32-bit**, select **X86**; if the system is **Windows 7**, choose **win7**; if it is **Windows 8**, choose **win8**; if it is **Windows 10 and above** , the choice of **Win8**.

9.Press **Next** .

10.You'll see a progress bar during the driver installation process.

11.After the driver installs, you'll get a notification letting you know the installation was successful—press **Close** to complete the process.

After completing the steps above, your device's driver is successfully updated.

You may need to restart your PC for any changes to take effect fully, however.

Resource Downloads

SDK

Current SDK version matches Robot version 3.4.3

Please download the SDK for your OS:

Windows
Linux

Firmware Image

Please download this **system image** to update your robot to version 3.4.3.

Sample Code

1. RobotMotion Sample Code: **RobotMotion**
2. RobotPlayer Sample Code: **RobotPlayer**
3. SpeechManager Sample Code: **SpeechManager**
4. RobotSystem Sample Code: **RobotSystem**

Avatar Studio

Avatar Studio: **AvatarStudio** | **User Manual**

Retail / Hospitality Application

Retail / Hospitality App: **Retail_app** | **User Manual**

Tools

1. AvatarMind Remote Controller(iRemoter):**iRemoter** | **User Manual**
2. AvatarMind Robot Update Tool(BurningTool)：**BurningTool** | **User Manual**
3. AvatarMind Robot Helper(iPalHelper): **iPalHelper** | **User Manual**
4. AvatarMind Robot Programmer(iPalProgrammer): **Windows** | **Linux** | **Mac** | **Android** | **User Manual**